A DIGITAL RECORDER
FOR POWER SYSTEM DISTURBANCES

A thesis submitted to the
Faculty of Graduate Studies
in partial fulfillment of the requirements
for the degree of Master of Science
in the Department of
Electrical Engineering

by
William Eric Norum
Saskatoon, Saskatchewan
January 1982

ACKNOWLEDGEMENTS

The author would like to thank Professors G.J. Huff and T.W. Clewes for their assistance, guidance and encouragement during the course of this project. The assistance of Mr. S. Hardy in the planning of the project and the efforts of Mr. R. Moffat during the field tests are gratefully acknowledged.

Financial support for this project was received from the National Research Council and from the Saskatchewan Power Corporation.

UNIVERSITY OF SASKATCHEWAN

Electrical Engineering Abstract 82A216

"A DIGITAL RECORDER

FOR POWER SYSTEM DISTURBANCES"

Student: Eric Norum

Supervisor: T.W. Clewes
Co-Supervisor: G.J. Huff

M. Sc. Thesis presented to the College of Graduate Studies

January 1982

ABSTRACT

This thesis describes the development of a prototype microprocessor-based fault recorder capable of digitally recording power system transient waveforms. The recorder can monitor and record 16 event channels and 8 analog channels with signal bandwidths of 350 Hz, or 16 event channels and 16 analog channels with signal bandwidths of 180 Hz. The recorder may be synchronized to an external satellite-controlled time base.

General information on the system hardware and software is presented. Results of laboratory tests, a description of field tests and recommendations for further development of the system are included.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

A/D	- Analog to Digital (conversion)
ALE	- Address Latch Enable
BCD	- Binary Coded Decimal
CPU	- Central Processing Unit
CSR	- Command and Status Register
CT	- Current Transformer
D/A	- Digital to Analog (conversion)
DEC	- Digital Equipment Corporation
EPROM	- Erasable Programmable Read-Only Memory
I/O	- Input/Output
IC	- Integrated Circuit
MODEM	- Modulator/Demodulator
PROM	- Programmable Read-Only Memory
PT	- Potential Transformer
p.u.	- Per Unit
RAM	- Random Access Memory
ROM	- Read-Only Memory
SPC	- Saskatchewan Power Corporation
TTL	- Transistor-Transistor Logic

1. Introduction

1.1 Overview

Advances in the field of digital electronics are bringing increasingly complex computer applications into the realm of economic feasibility. Digital processors are simultaneously and rapidly becoming more powerful and less expensive. This project is the application of the current level of microcomputer technology to create a research tool in the area of power system fault recording. The application presently requires all of the power of the processor to give performance in the desired range. However, the rapid advance of the state of the art in mini and microprocessors promises to make future versions of the recorder not only more powerful, but also less expensive.

In power system switching stations, instrumentation is required to monitor, control and record quantities associated with power system operation. Traditionally the recording system has consisted of an analog strip chart recorder writing on light-sensitive paper. Difficulties with this method range from the reliability and maintainability of any precision electro-mechanical device to the problems of converting the data to a form in which it may be processed and analyzed by computers. The latter point is of particular importance since if it were routinely possible to transfer fault information to a central computer it would provide an

easy means for system planners to ensure that their system simulations were indeed accurate.

The Saskatchewan Power Corporation, recognizing the advantages of digital techniques, included fault recording as a high priority in their "Switching Station Automation" project. The goal of that project was to perform all switching station instrumentation on a computerized system. A single, rather large, minicomputer was used as a base for the system. The success of the project lead to the project described in this thesis; namely the feasibility of performing the fault recording function on a microcomputer.

1.2 Design Objectives

The following design objectives were developed in consultation with the author's thesis supervisors and representatives from the Saskatchewan Power Corporation [5]. The reasons behind the objectives are described in considerable detail in the final report of the Saskatchewan Power Corporation switching station automation project [3].

Intended Use

The digital fault recorder is to be a portable instrument capable of digitizing and recording samples of voltage and current waveforms supplied by conventional potential and current transformers. The recorder is intended to be installed in switching stations or substations on a tem-

porary basis. The intended uses of the recording system are to provide records of fault data (waveform reconstruction) and to provide data for the calculations of power flows during system disturbances (load shedding tests, system stability studies).

Sampling and Storage Capacity

Carr et al[2] have indicated that the minimum sampling rate for reasonable waveform reconstruction is 240 Hz. A minimum of 7 channels are required to monitor the 4 currents and 3 voltages of a single transmission line. A higher sampling rate and larger number of channels are desirable. At least 16 channels of digital signals must also be sampled and recorded. The recorder should provide at least 2 seconds of prefault and 120 seconds of postfault information. Some indication of the amount of recording space remaining should be provided. The recording time should be adjustable and should be dependent on the continued presence of a system disturbance. Thus, a recording will have a minimum duration which will be exceeded if the disturbance which initiated the data acquisition persists for longer than the specified recording duration.

Triggering

Data acquisition may be triggered by any selected channel or channels. Triggering can be set to occur when the rate of change of the signal exceeds a preset value. The

trigger thresholds must be easily adjustable. Triggering will also be possible from an external contact closure or change of state on a digital input.

Timing Options

The digital fault recorder will incorporate a real-time-clock to set the sampling rate and to maintain the date and time. Provision must be made for attaching an external time signal (such as a satellite-controlled clock) to the recorder. The satellite clock option allows data from geographically remote recorders to be compared and analyzed on the same time base.

Accuracy and Range

Voltage signals should be measured to an accuracy of Ø.2% of 1 p.u.. Overvoltages of 4 p.u. should be recorded without limiting.

Current signals should be measured to an accuracy of 1.Ø% of 1 p.u.. Overcurrents of 2Ø p.u. should be recorded without limiting.

These objectives reflect the quantization inherent in digital recording. There is a direct tradeoff between the 1 p.u. accuracy and the maximum signal to be recorded without limiting (assuming a linear analog to digital converter).

Interfacing Requirements

The microcomputer hardware will have analog inputs capable of measuring input signals from -10 to +10 volts. Digital inputs will be TTL compatible. This project deals only with the hardware associated with the microprocessor so signal isolation and conditioning will be supplied by SPC to convert the signals from standard PT's and CT's to the desired range.

Interaction with the system will be through a standard serial line which may be connected to a hardcopy or video terminal, or through a modem to the telephone network. Ontario Hydro has shown the advantages of being able to remotely access recording devices[4]. Data transfer from the storage medium will be possible on the serial line although this will only be practical for transmitting short records.

Packaging Requirements

The system should be packaged in a portable enclosure suitable for use in a switching station (indoor) environment. The system must be capable of operating in an electrically noisy environment. Sufficient electrical shielding and filtering must be provided to prevent this noise from interfering with the operation of the unit.

The system must be portable and general purpose since a specific test site was not chosen.

Power Calculation

Another objective that received some discussion was the real-time calculation of power flows in the Regina South minicomputer installation. This calculation was felt to be desirable since it would allow triggering from power flow disturbances as well as from current or voltage disturbances. In addition, if only the power flow signal was recorded, a substantial reduction in the amount of data to be recorded would be achieved.

This objective was discarded early in the development of the recorder since no available microcomputer was powerful enough to perform the required calculations.

2. Digital Fault Recorder Hardware

2.1 Introduction

Once the recorder design goals were specified, possible methods of implementing the system were considered. The first two sections of this chapter outline the major decisions made regarding the recorder hardware. Following this, a description of the hardware elements of the recorder is presented. Hardware constructed specifically for this project is discussed in more detail than hardware which was simply purchased.

2.2 Microprocessor Selection

The choice of microprocessor is critical as it affects almost all the subsequent hardware and software aspects of the system. The points which must be considered in selecting a microprocessor for a particular application include power consumption, speed, instruction set, availability of peripherals and cost. An additional point is ease of programmability; that is, once the hardware has been selected, how much difficulty will there be in writing, testing, and debugging the software. Because the fault recorder was to operate in the switching station, power consumption was not considered a factor in processor choice for this application.

The processors widely available at the time that this

choice was being made were the Intel 8080, the Motorola 6800, and the Digital Equipment Corporation LSI-11. These processors are described in detail in references 13, 18 and 15 respectively. All three were available as packaged systems including case, power supply, card-cage, and backplane. Similar peripherals were available at about the same cost for all three. All three processors were felt to be fast enough to meet the recorder requirements. The 8080 and the 6800 are 8-bit microprocessors and the LSI-11 is a 16-bit microprocessor. The 16-bit word length of the LSI-11 was considered an advantage for dealing with the data from the 12-bit analog to digital converter. The LSI-11 can operate directly on floating point numbers, an ability not shared by the other two microprocessors. This is an important feature for this application because of the large amount of floating point arithmetic done in the waveform reconstruction portion of the recorder.

The factor which most influenced the selection of microprocessor was that of programmability. Use of either the 6800 or the 8080 would have necessitated writing all of the software in assembly language. Higher level languages existed for these processors but could not be used as they were either too slow, or required the purchase of an expensive development system from the manufacturer. Code for the LSI-11 could be written and tested on the Electrical Engineering PDP-11/60 and then loaded into the LSI-11 for execution. In addition, software for the LSI-11 could be

written either in assembly language, or in a variety of high-level languages available on the larger members of the PDP-11 family.

After consideration of the above factors the LSI-11 was chosen as the microprocessor for use in the digital fault recorder.

2.3 Recording Device Selection

The second major hardware component in the digital fault recorder is the device upon which the data will ultimately be recorded. The devices available to interface to the LSI-11 were magnetic tape, both cassette and standard 9-track, and magnetic disk, both hard and floppy.

The cassette tape system was eliminated from consideration immediately as the recording rate was much too slow to allow continuous recording of fault data.

Nine-track magnetic tape was considered to be advantageous for several reasons. Large quantities of data could be stored on a single tape and recording rates are high. The tapes could be read and used by almost any other computer system. The major shortcomings of mag-tape, which precluded its use in the digital fault recorder, were the high cost and large size and weight of the tape transport. In addition, the data on the tape is not block replaceable. This means that data may be written only at the end of data

already on the tape, so any unwanted data could not be overwritten without overwriting the entire tape.

Hard disks share the advantages of high recording rate and large data storage volume with nine-track mag-tape. Unfortunately, the only drives available for use with the LSI-11 used removable cartridges, as sealed "Winchester" drives were not yet available. The removable-cartridge drives were heavy, expensive, fragile, and required a clean environment in which to operate. This would have defeated the portable nature of the recorder, and so hard disks were dropped from consideration.

This leaves only the floppy disk as a recording medium. Floppy disks suffer from limited storage capacity and recording rates, but have several advantages for this application. They are small, lightweight, quite rugged and inexpensive, all of which are factors critical to the digital fault recorder. They share with mag-tape the advantage of industry-wide compatibility. A more detailed description of the floppy disk system selected for the digital fault recorder is presented in a following section.

2.4 Hardware Description

As described in the previous sections, the digital fault recorder is based on an LSI-11 microprocessor and a floppy disk storage system. A detailed description of the LSI-11 and peripheral equipment is provided in references 14

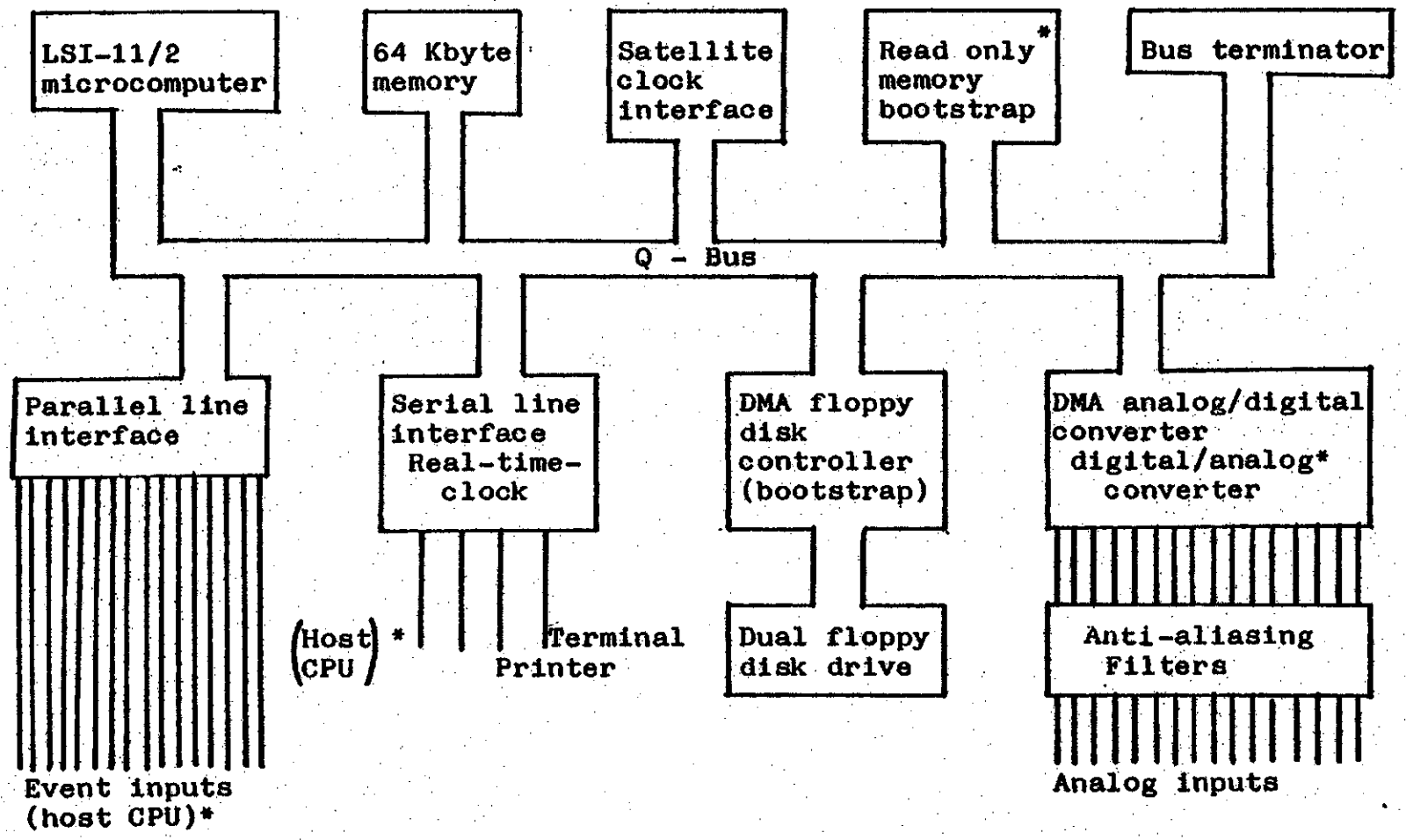
and 15. The overall structure of the hardware is shown in block diagram form in figure 2.1. The items marked with an star (*) were used only during development of the digital fault recorder, and are not present in the final unit. The following sections describe each of the hardware elements in further detail.

2.5 Processor

The LSI-11 is the smallest member of the Digital Equipment Corporation PDP-11 product line. It is a 16-bit processor with fixed, and floating, point arithmetic capabilities. Direct memory access allows peripherals to transfer data to or from memory without affecting processor operation. The most severe limitations imposed by the LSI-11 are its ability to access only 64k bytes of memory, and the presence of only a single level of device interrupts. The LSI-11 is contained on a single 5.25 by 8.9 inch (dual size) board, and communicates with the other hardware modules via a 38 line bus. The backplane used in the digital fault recorder has space for 16 dual, or 8 quad size boards. The backplane is mounted in a 5.25 inch high standard 19 inch rack-mount box along with power supplies, fans, and a control panel.

2.6 Memory

This module uses 32 16k bit dynamic memory chips to



* Present during development only

Figure 2.1 Digital Fault Recorder Hardware Block Diagram

provide 32k 16-bit words of randomly addressable memory. Due to the memory-mapped-i/o architecture of the LSI-11 only 30k of this memory could be used. All the data taken from the analog and event inputs are buffered in this memory to provide prefault recording capability and to prevent the disk heads from being continuously in contact with the floppy disk.

2.7 Serial Line Interface

The serial line interface provides 4 RS-232 serial lines for communication between the digital fault recorder and a console terminal, a printer, and other computers. It also generates the timing signal which fixes the sampling rate of the recorder. The use of one of the channels to provide this real-time-clock facility is described in the satellite clock interface section. The data rates of the remaining three channels may be individually adjusted from 150 to 38400 bits per second.

2.8 Floppy Disk System

A DSD-440 dual drive, single sided, full size, double density floppy disk system is used in the digital fault recorder. The DSD-440 is a complete emulation of the DEC RX02 floppy disk system. Each diskette has a storage capacity of 512,512 bytes. The drives and controller are mounted in a separate 5.25 inch rack mountable enclosure.

The controller interface is contained on a single, dual size, board and is connected to the drive subsystem by a 26 pin ribbon cable. The controller interface uses direct memory access to transfer data to and from the digital fault recorder's memory. This is the most sophisticated form of input/output available in an LSI-11 system. It has the advantages of high speed and little or no program interaction after the initial start-up. The reduction in processor load provided by DMA disk I/O is critical to the operation of the digital fault recorder. If the disk did not provide DMA I/O the LSI-11 would not have time to service it and still perform the sampling and triggering functions.

The maximum sustained data transfer rate to the floppy disk is about 16000 bytes per second. This proved to be the factor which limited the sampling rate, and number of channels, of the digital fault recorder.

The floppy disk controller interface board also contains a bootstrap program stored in non-volatile read-only-memory. On power-up, this program reads the digital fault recorder software off the disk into memory, and then begins execution.

2.9 Analog to Digital Converter

The analog to digital converter samples the analog signals from the switching station PT's and CT's and converts them to a digital representation. A Data Translation DT1761

converter is used in the digital fault recorder. This a/d converter module provides 12 bit conversion accuracy on each of 16 channels and is contained on a single quad size board. A feature of prime importance is that the DT1761 provides DMA transfer of the digitized data to the digital fault recorder memory. The reduction in processor load provided by DMA operation of the a/d converter is critical to the operation of the recorder.

The DT1761 may optionally be configured to include 2 analog output channels. Such a board was used during the development of the digital fault recorder.

2.10 Anti-Aliasing Filters

There are two separate sets of filters in the digital fault recorder. One set, with a cutoff frequency of 180 Hz is used when the recorder is operating at its 480 Hz sampling frequency. The other set, with a 350 Hz cutoff is used when the recorder is sampling at 960 Hz. All the filters are active second-order Butterworth constructed around LM124 quad op-amps [9]. The filter design, and schematics are included in appendix 6.1.

The filters are mounted in a 3.5 inch rack mountable enclosure and are connected to the DT1761 a/d board by a 50 pin ribbon cable. All the analog and event inputs of the digital fault recorder are mounted on the front of this enclosure.

2.11 Parallel Line Interface

This dual size board provides the 16 event inputs to the digital fault recorder. It is a Data Translation DT2768 which is equivalent to the DEC DRV11. All inputs and outputs are TTL compatible. The recorder places a status word on the 16 output lines of this interface. Other devices may use this status word to monitor the operation of the recorder. During development of the recorder software this parallel line interface was used as a high speed link between the recorder and a host computer.

2.12 Satellite Clock Interface

The satellite clock interface took two forms during the development of the digital fault recorder. The first interface was a custom constructed board which allowed direct connection of the recorder to an Arbiter Systems satellite clock receiver [16]. The second interface consisted of two parallel line interfaces identical to that described in the previous section. This version accepted timing signals in a form that SPC has standardized on. These interfaces are described in the following two subsections.

2.12.1 Custom Satellite Clock Interface

As mentioned in the previous paragraph, this version of the satellite clock interface was constructed especially for the digital fault recorder. The interface requirements were

felt to be complex enough to warrant use of a microprocessor. The interface was constructed on a dual-width wire-wrap module and was based on an Intel 8748 single chip microprocessor [12]. The 8748 was chosen as it was a completely self-contained device, with on-chip I/O , timer, RAM, and EPROM.

The software for the 8748 was written in assembly language and assembled using a cross-assembler running under the UNIX operating system [7]. The program was burned into the 8748 EPROM using an EPROM programmer connected to the PDP-11.

The first function performed by the satellite clock interface was to generate an accurate timing signal to set the sampling rate of the digital fault recorder. The hardware for this function is shown in figure 2.2. The IC numbers refer to the position of the IC on the interface board. The Address Latch Enable (ALE) output of the 8748 is a TTL level signal which operates at 1/15 of the clock frequency of the 8748. A 6 MHz crystal was used to provide the timing reference for the 8748. The 400 kHz ALE signal is passed through rate generator IC 5F and emerges as a 120 kHz signal which is fed through three divide-by-5 IC's (4F, 3F, 2F) to emerge as a 960 Hz signal. This signal is then passed back through other portions of 2F, and 3F to provide 480 Hz and 240 Hz signals. All three clock signals are sent to the anti-aliasing filter board which selects one of them

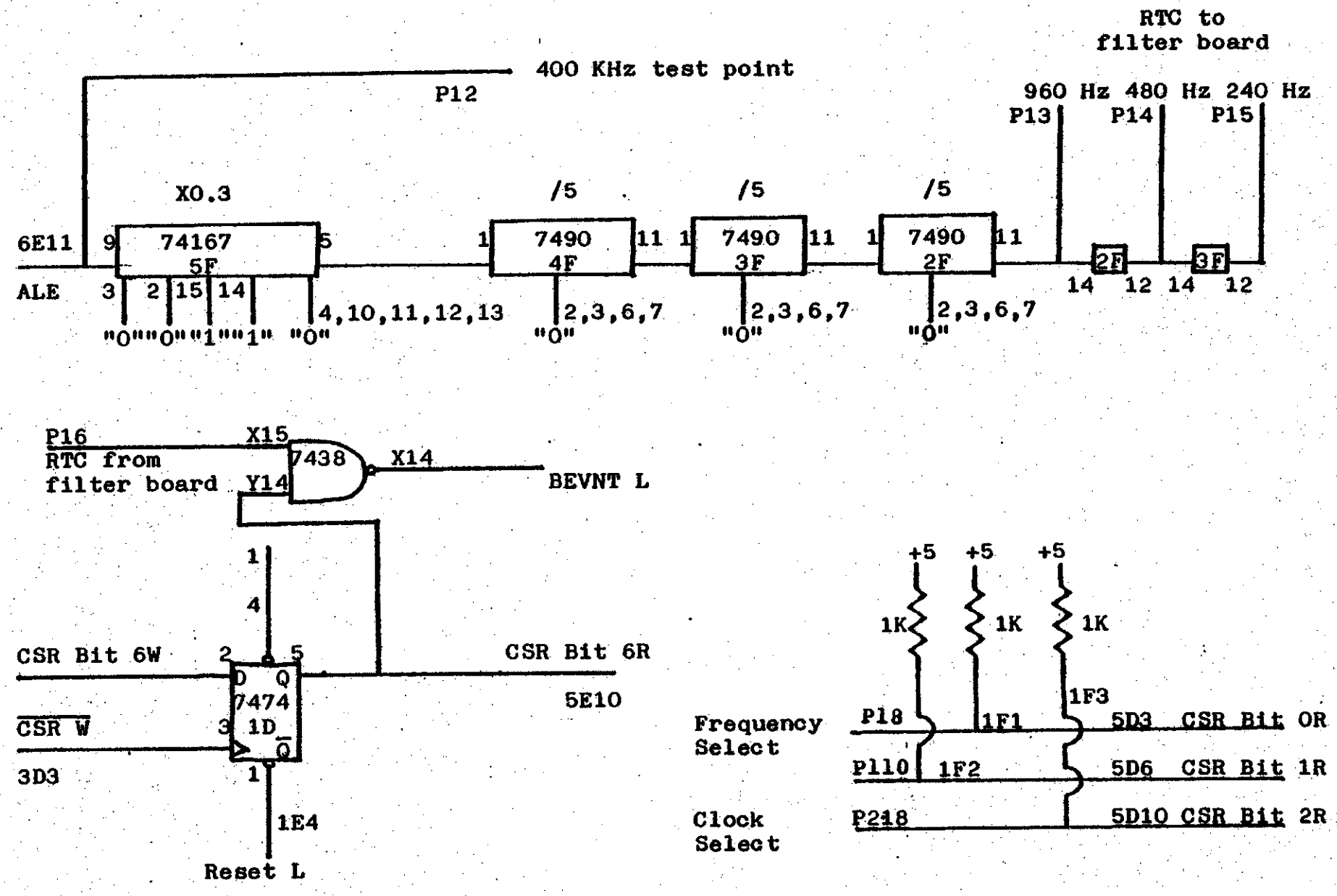


Figure 2.2 Real-Time-Clock Interrupt Generation

and returns it to the satellite clock interface board. Thus the sampling rate is set automatically by the selection of an anti-aliasing cutoff frequency. The clock signal is gated by CSR bit 6 (real-time-clock interrupt enable) and then passed to the bus event line. Pulling this line low causes the LSI-11 to perform an interrupt service through address 0100_8. Use of the bus event line to generate interrupts results in a considerable simplification of the satellite clock interface as no interrupt acknowledge or vector generation hardware need be provided. In order that the LSI-11 can determine which clock frequency has been selected, the filter board also controls bits 0 and 1 in the CSR. The state of these bits is interpreted by the LSI-11 as shown in table 2.1.

Bit 1	Bit 0	Interpretation
0	0	240 Hz clock
0	1	480 Hz clock
1	0	960 Hz clock
1	1	No filter board

Table 2.1 Clock selection bit interpretation

In a like manner, CSR bit 2 is controlled by the satellite clock receiver. It is high when no receiver is connected and is pulled low when a receiver is connected.

The second function performed by the satellite clock interface was to provide time-of-day information to the LSI-11 so that the time and date of recordings may be noted. This portion of the satellite clock interface is shown in

figure 2.3. The LSI-11 indicates that it wants to read the time by writing a 1 into bit 4 of the CSR. This raises P2-3 in the 8748 which causes it to copy the current time and date to a buffer and then send the contents of that buffer to the LSI-11 a byte at a time. First, three bits of label information are latched out to P2-Ø through P2-2. The time data corresponding to this label is then latched out on the 8748 bus lines which also generates a pulse on the $\overline{WR}$ line. This raises bit 7 in the LSI-11 CSR and pin TØ in the 8748. When the LSI-11 sees CSR bit 7 go high it reads the time label and data from the data register. This resets CSR bit 7 and 8748 pin TØ. When the 8748 sees pin TØ go low it sends the next label and data byte. This process is repeated until all the time and date information has been transferred to the LSI-11. This information consists of 5 BCD bytes indicating the time and date at which the clock was last set, and 5 bytes indicating how many clock ticks have occurred since that time. The 8748 timer "ticks" 25ØØ times per second and thus can time intervals of $\frac{2^{4Ø}}{25ØØ}$ seconds (over 13 years) before requiring resetting.

The remaining hardware on the satellite clock interface board was used to set the clock in the 8748. The set time signals may come from either the satellite clock receiver or from the LSI-11. If the receiver is connected to the interface it will set the 8748 clock once every second. If no receiver is connected the 8748 clock is set from the LSI-11 under operator control. Operation of the 8748 is identical

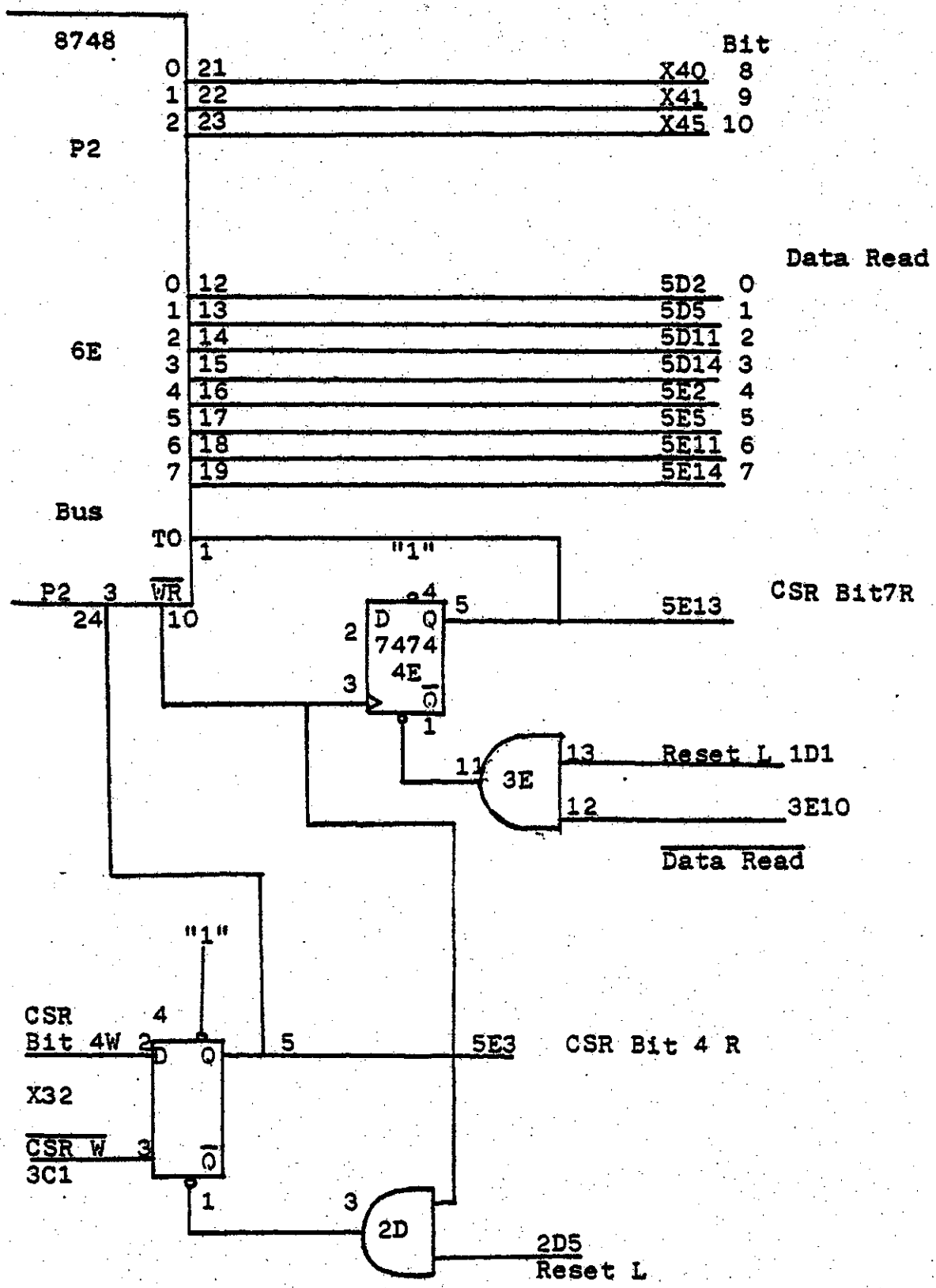


Figure 2.3 Time Data to LSI - 11

in both cases, the only difference is the source of the set time signals.

A listing of the 8748 program is contained in appendix 6.7. The program is short, and self-explanatory, so no discussion of the program will be presented here.

During final testing of the 8748 satellite clock interface SPC indicated that a standard clock interface had been chosen and that it would be preferable that the digital fault recorder conform to this standard. Work on the 8748 version of the satellite clock interface was halted and design of a new interface was begun. A description of this interface is provided in the following section.

2.12.2 Parallel Satellite Clock Interface

The SPC clock interface standard has 30 data lines and a single control line. The data lines are assigned as shown in figure 2.4.

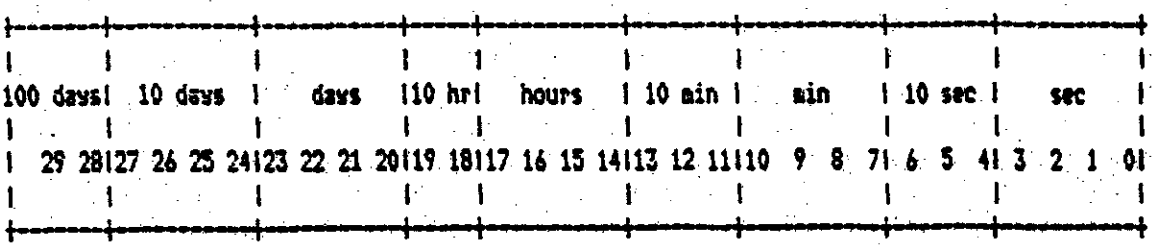


Figure 2.4 SPC standard clock interface bit definitions

The rising edge of the control signal indicates that the

time and date information on the data lines is valid.

It was very easy to adapt the digital fault recorder to conform to this standard, as all that was required was to add two DT2768 parallel I/O boards to the system. The low order 16 bits of the time and date are connected to one board, and the high order 14 bits are connected to the other. The remaining 2 bits of the second board are not used.

The real-time-clock generation function of the 8748 satellite clock interface was not provided by this parallel interface technique so some other means of generation had to be provided. A commercial real-time-clock board for the LSI-11 could not be used as all the available units were based on a 1 MHz clock and could not generate the required 960 Hz interrupts. It was discovered that the crystal controlled serial-line rate generator could be used as an accurate source of 960 Hz interrupts.

One of the unused serial line ports was designated as the "real-time-clock" port. This port was then set for 9600 baud, 8 bit data transmission with no parity, and a single stop bit. Thus 10 bits are transmitted in a serial fashion for each byte loaded into the transmitter buffer. Because the serial line interface uses double buffering a byte may be loaded into the interface while the immediately preceding byte is being transmitted. If the byte is loaded before the preceding byte has been completely sent the transmitter

ready interrupts will occur at a fixed rate of 960 Hz. Using every other interrupt to start a conversion results in a 480 Hz sampling rate.

The LSI-11 counts the clock interrupts to maintain the time and date if no satellite clock receiver is present, and to provide fault time resolution better than that provided by the satellite clock receiver.

2.13 Packaging

The computer, disk drives, and anti-aliasing filters are each contained in a 19 inch rack mountable chassis. The three units require a total of 14 inches of (vertical) rack space and were mounted in a 4 foot high cabinet. The rest of the cabinet space was empty. The recorder weighs about 55 kg when mounted in the cabinet. The disk subsystem requires that the ambient temperature be between 15 and 35 degrees Celsius [14]. The computer and filter subsystems are less restrictive, requiring only that the temperature be between 5 and 40 degrees Celsius [15].

3. Digital Fault Recorder Software

3.1 Introduction

This chapter provides a general description of the main software modules of the system, and their interaction. A detailed discussion of every software module is not presented as the comments in the listing provide sufficient explanation of the purpose and operation of the module. A full listing of the digital fault recorder software is included in appendix 6.6. A command manual for the recorder is included in appendix 6.5. A description of the software development system on which the software was written will be presented before the recorder software is discussed. In the discussion of the recorder software variable names and routine names have been underlined.

3.2 Software Development System

The digital fault recorder program is about 24k bytes long and is written almost entirely in the programming language C. C is a modern, general-purpose programming language and contains facilities to ease the development of well structured programs [6]. These facilities include: statement grouping; decision making (if, else); looping (while, for, do); and selecting one of a set of possible cases (switch).

The major features of the software development system,

shown in block diagram form in figure 3.1, are described below.

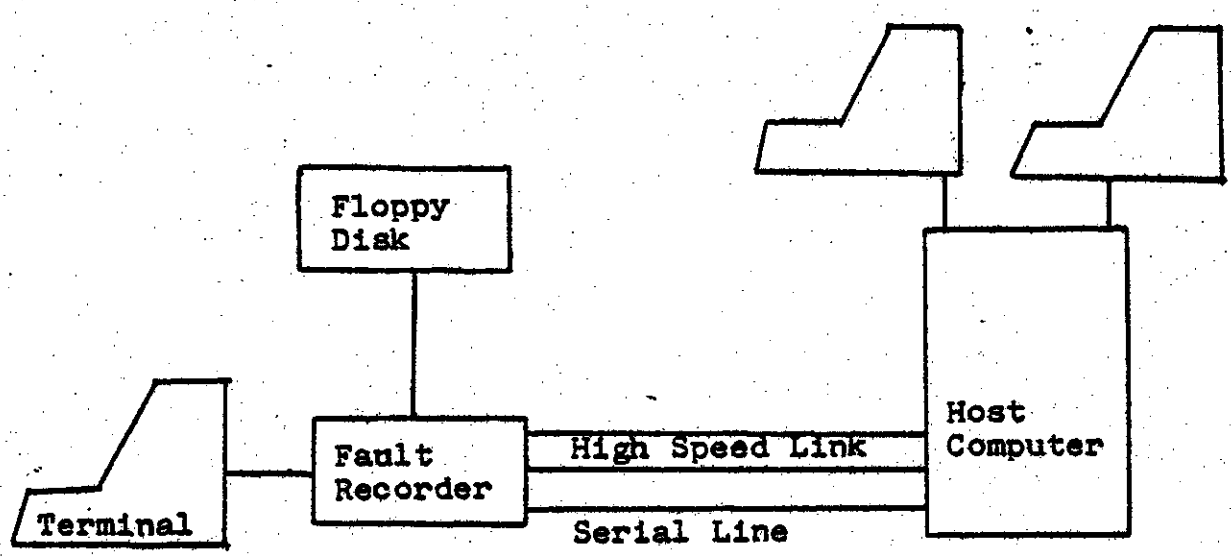


Figure 3.1 Software development system

Upon power-up, the LSI-11 runs a program that transfers data from one serial line to the other. Thus, the terminal appears to be directly connected to the host computer. Programs are written and compiled on the host computer using all the tools available under the large operating system. When the program is to be executed the system in the host computer determines that the program must be executed remotely, and then loads the program into the LSI-11. The terminal now appears to be communicating with the program in the LSI-11. When this program terminates, the LSI-11 begins transferring data between the serial lines again, and the the terminal once again appears to be connected to the host computer. The quick turnaround provided by this system contributed much to the ease of writing the fault recorder

software. The development system is similar to that described by Lycklama and Christensen[10].

3.3 Data Structures

An understanding of the data structures used within the digital fault recorder is necessary before the discussion of the software modules that manipulate this data can begin. In the following sections the term 'analog data' is used to refer to data that have been taken from the a/d converter, and the term 'digital data' is used to refer to data that have been taken from the event inputs.

The fundamental element of data manipulated by the fault recorder is a sample of analog and digital data. In order to provide maximum data storage and prefault recording time, the analog and digital data are packed together as shown in figure 3.2. The last 8 elements are not present if only 8 channels of analog data are being taken (960 Hz sampling rate). Thus, at the 960 Hz sampling rate a sample block is 16 bytes long, while at the 480 Hz sampling rate a sample block is 32 bytes long. The data rate is the same in both cases; 15360 bytes per second. The sample number is maintained as a check that data on the disk have not been corrupted. In order to reduce the number of bit clearing and setting operations the sample number, and event channel data, are stored as the 1's complement of their actual value. The timing flag (f), if zero, indicates that the

Word	Bit in Word 15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	e15	e14	e13	e12	Analog Channel 0											
1	e7	e6	e5	e4	Analog Channel 1											
2	e3	e2	e1	e0	Analog Channel 2											
3	e11	e10	e9	e8	Analog Channel 3											
4	sn3	sn2	sn1	sn0	Analog Channel 4											
5	f	1	1	1	Analog Channel 5											
6	1	1	1	1	Analog Channel 6											
7	1	1	1	1	Analog Channel 7											
8	1	1	1	1	Analog Channel 8											
9	1	1	1	1	Analog Channel 9											
10	1	1	1	1	Analog Channel 10											
11	1	1	1	1	Analog Channel 11											
12	1	1	1	1	Analog Channel 12											
13	1	1	1	1	Analog Channel 13											
14	1	1	1	1	Analog Channel 14											
15	1	1	1	1	Analog Channel 15											

Figure 3.2 Single sample data block

time and date associated with the fault refer to this sample in particular. The analog data are stored as offset binary integers, with 0 corresponding to -10 volts and 7777_8 corresponding to +9.995 volts.

The sample blocks are written contiguously on disk number 1 when a fault is detected. Each disk block holds 256 bytes, which corresponds to 1/60'th of a second of data, or one power line cycle. Thus the term 'cycle' is used as a synonym for a disk block in the fault recorder software.

The digital fault recorder maintains a set of directory entries so that data for a particular fault may be accessed easily. Each directory entry contains all the parameters

for a particular fault. These parameters include :

a) the disk addresses of the first and last block of recorded data for this fault.

b) the type of clock (internal or external) in use when the fault occurred.

c) the number of analog channels, the sampling rate, and the number of samples per power-line cycle.

d) the time and date of the fault.

e) the recording label.

f) the event channel parameters.

g) the analog channel parameters.

h) the number of pre, and post, fault blocks.

The event channel parameters consist of a channel name and two flags indicating the triggering conditions for the channel. There are 16 such parameter structures, one for each event channel.

The analog channel parameters consist of a channel name and a triggering level. There are 16 such parameter structures, one for each analog channel. The latter 8 structures are not used if only 8 analog channels are being sampled.

Directory entries are contained on the last half of the disk in drive Ø. An extra directory entry not associated

with any fault is used to store parameters so that they may be read from disk and thus not have to be typed in each time the fault recorder is powered-up.

The layout of the data on the disks is shown in figure 3.3. Empty directory slots in the middle of the directory table, and empty data areas between other data are automatically reclaimed by the fault recorder each time the start command is given, by moving the latter blocks down over the empty spaces. If the data on disk 1 are moved, the corresponding directory entries are updated to reflect the changed position of the data.

3.4 Data Acquisition and Triggering

The heart of the digital fault recorder consists of the routines which acquire data and watch for fault conditions. These routines are interrupt driven, as shown in figure 3.4. Because they are so critical to the operation of the recorder a complete description of these routines will be presented.

The primary routine is the real-time-clock interrupt service routine (rtcint). The clock runs continuously, regardless of whether or not data are being acquired. The actions of the routine are shown in flowchart form in figure 3.5. First the 'real-time-clock' is reprimed by writing into the serial line interface transmitter register. The time-of-day counter is then incremented and the sampling

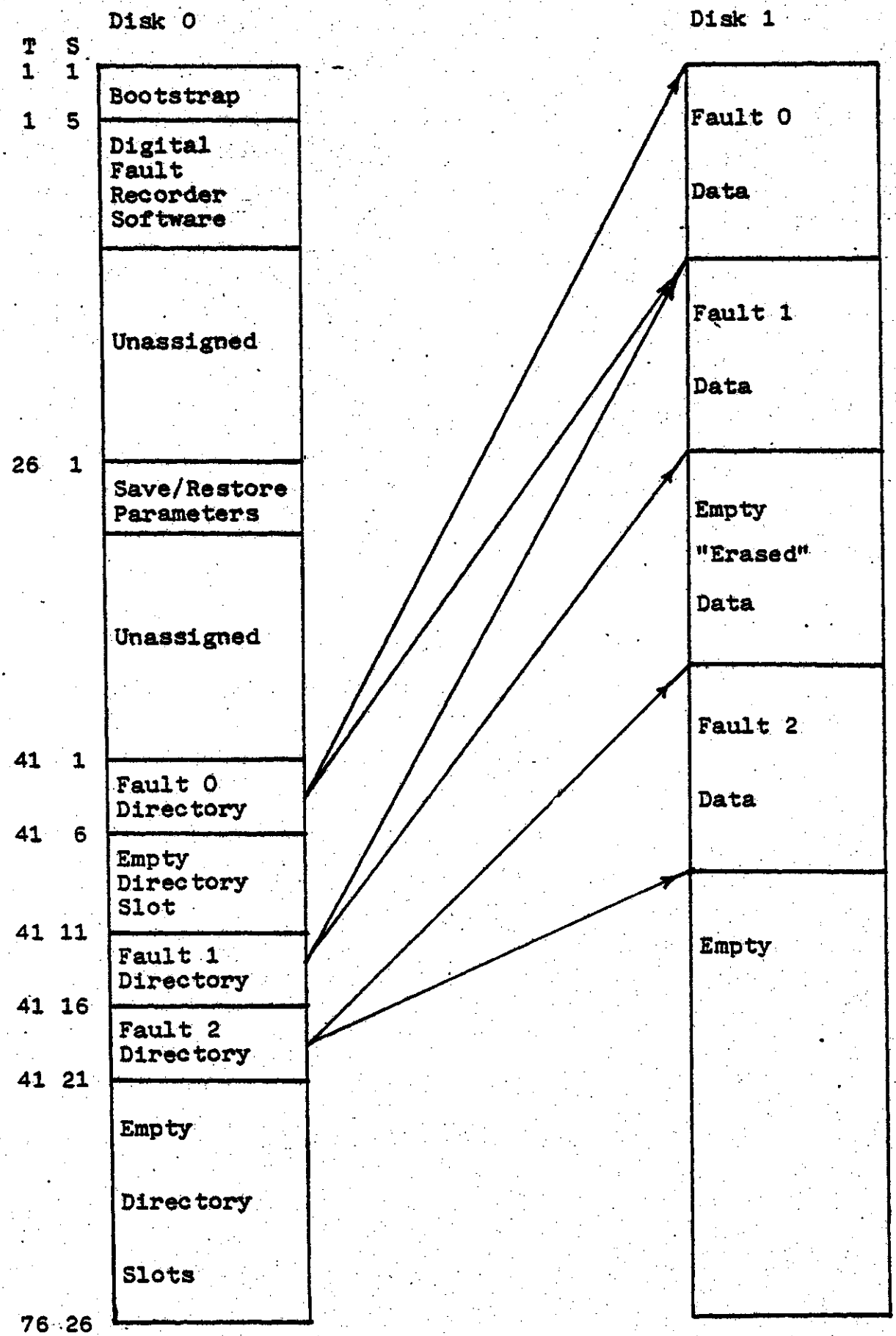


Figure 3.3 Disk Layout

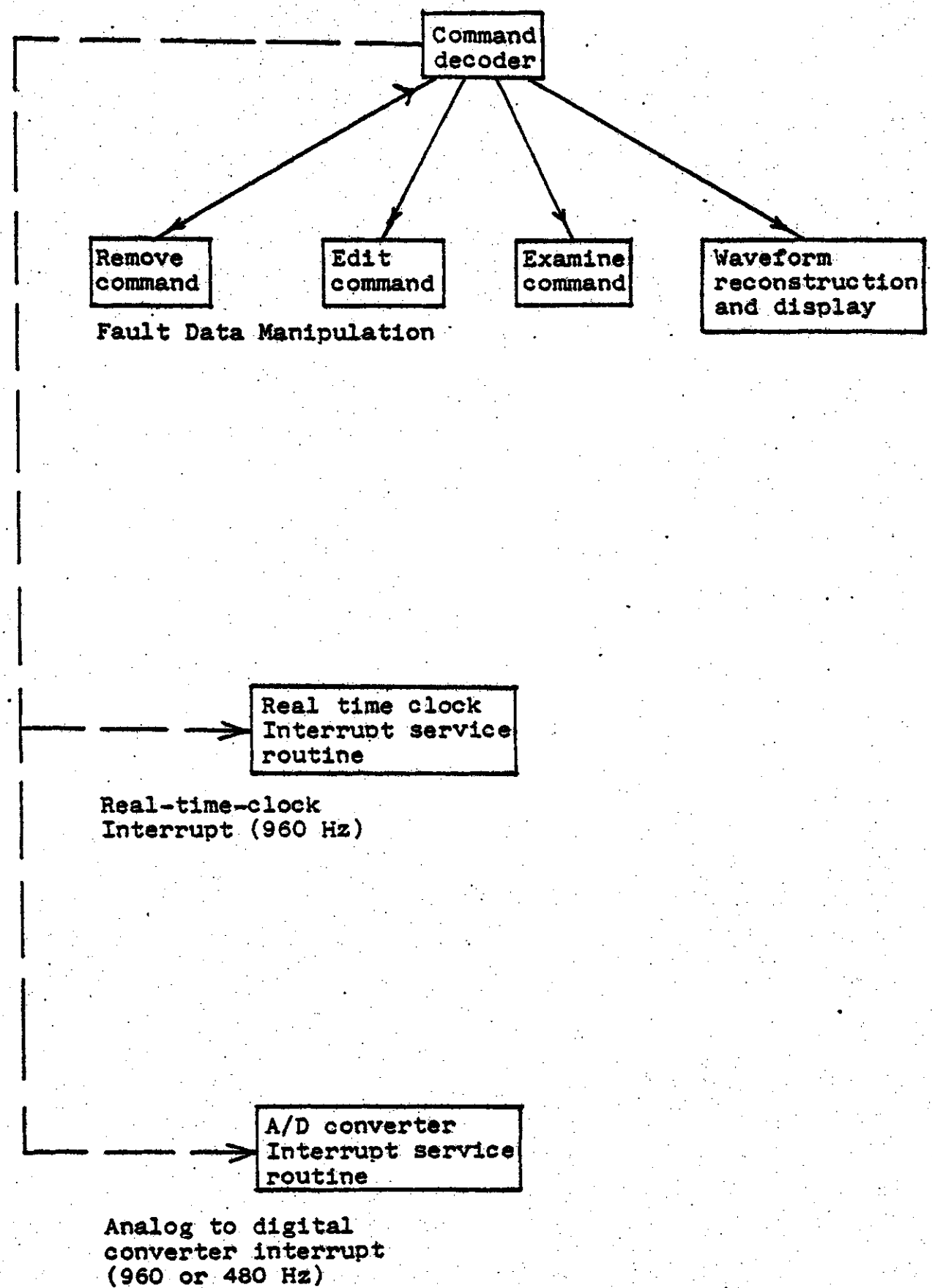


Figure 3.4 Software Block Diagram

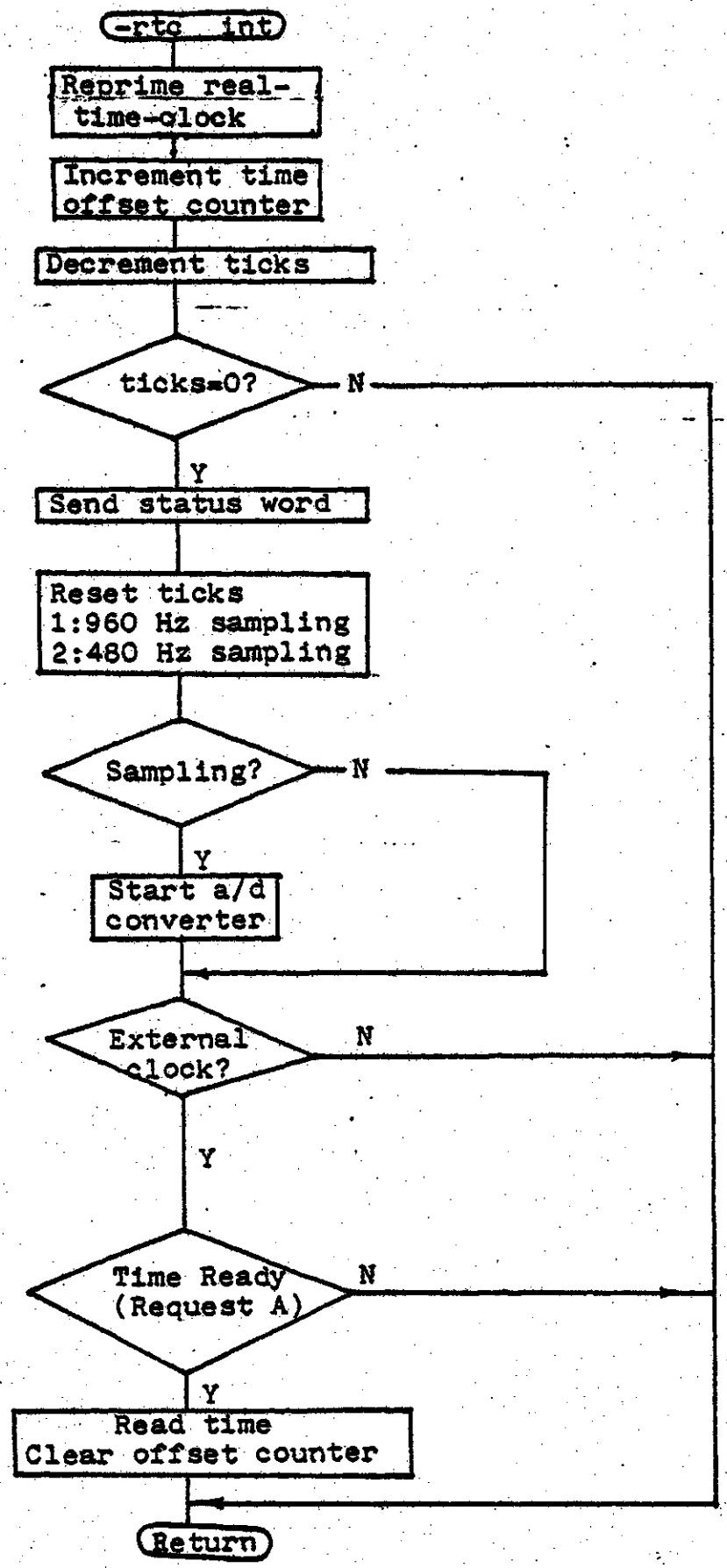


Figure 3.5 Real Time Clock Interrupt Service Routine

tick counter decremented. If the sampling tick counter is not yet zero the interrupt service routine returns. If the counter has reached zero it is reset and the system status word is sent to the parallel (event) interface output channel. If the system status word indicates that sampling is to be done (Bit Ø Hi) the a/d converter is started. If there is no external time-of-day clock connected to the recorder the interrupt service routine returns. If there is an external clock, and if it has valid data, the time and date are read and the time-of-day counter is set to Ø. The interrupt service routine then returns. Note that the time-of-day counter is updated even when an external clock is connected. Thus time and date information is provided with the accuracy of the external clock, and with the resolution of a single sampling interval. Data from two, remote, recorders can be synchronized to within a single sample. The time-of-day counter is a 32 bit counter, so the time and date must be set at least once every $\frac{2^{32}}{960}$ seconds (about 51 days) if an external clock is not connected to the recorder.

The a/d converter interrupt service routine (_monitor_), which implements the triggering algorithm, is shown in flowchart form in figure 3.6. This routine is entered when the a/d converter finishes sampling the analog channels and storing the corresponding digital data in memory. Its first action is to check that it has not been reentered, that is, that the previous invocation of the routine has exited. If

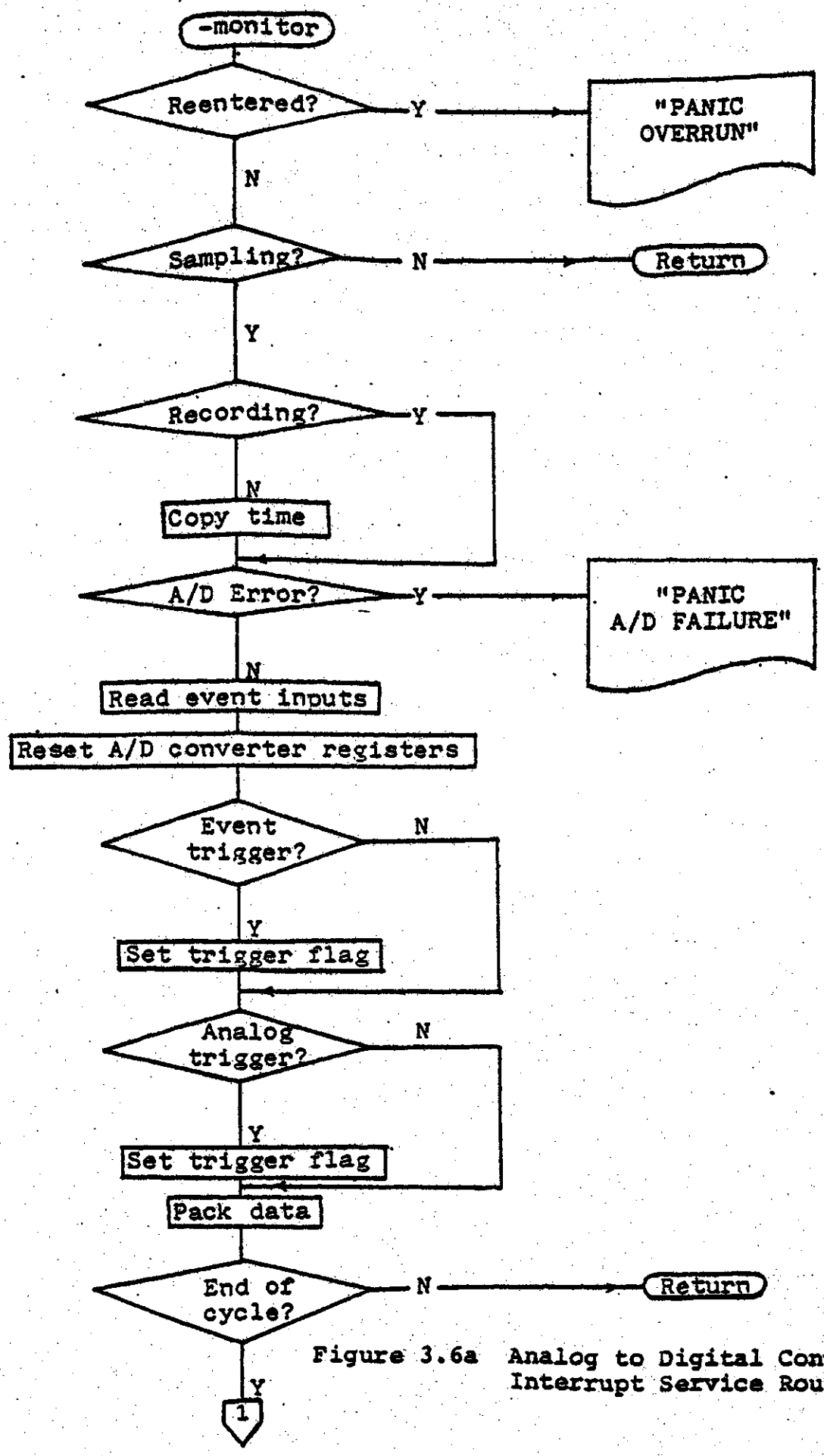


Figure 3.6a Analog to Digital Converter Interrupt Service Routine

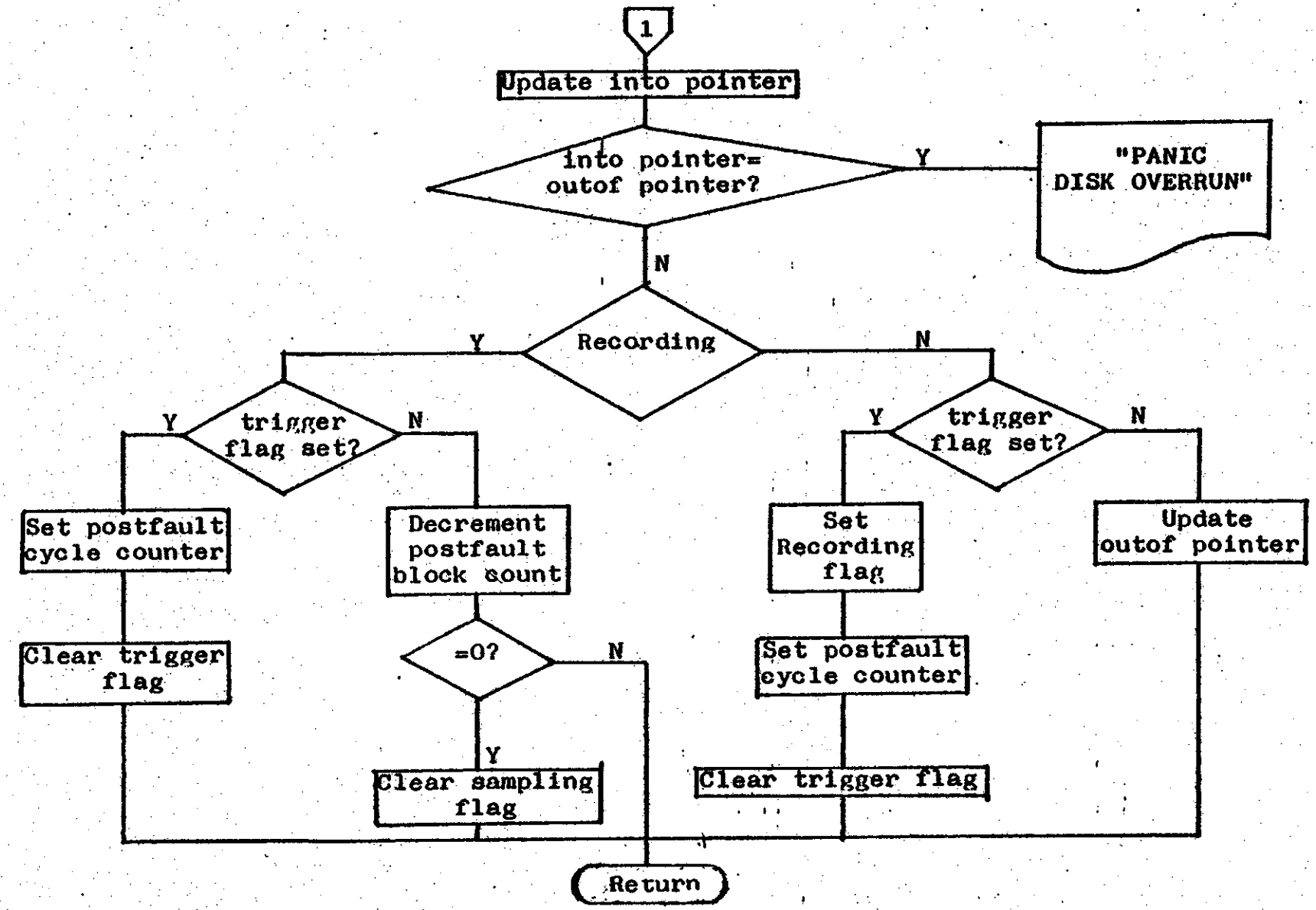


Figure 3.6b Analog to Digital Converter Interrupt Service Routine

the routine has been reentered an error message is printed on the console and the program halts. If the routine has not been reentered it then checks the state of the sampling bit in the system status word. The monitor routine exits if this bit is not high. If the bit is high, the state of the recording bit in the system status word is tested. The current time and date information is copied if this bit is off. Thus, when a fault is recorded this copy indicates the time and date at which the fault was detected. A check is then made for a/d converter hardware errors. If an error has occurred, a diagnostic message is printed and the program halts. If an error has not occurred, the event inputs are read and the a/d converter DMA control registers are reset. The a/d converter always acquires data from all 16 channels even if the recorder is sampling at 960 Hz and using only the first 8 channels. The memory into which the a/d converter places the data is arranged as a ring buffer, shown schematically in figure 3.7. The a/d pointer points to where the data for the next set of analog data will be DMA'd into. The function of the other pointers to this buffer are described later in this section.

The monitor routine then checks to see if an event trigger has occurred. First, a check is made to see if any of the event lines have changed state since they were last sampled. This operation is performed by taking the exclusive-or of the event state and the state saved by the previous invocation of the monitor routine. If no bits

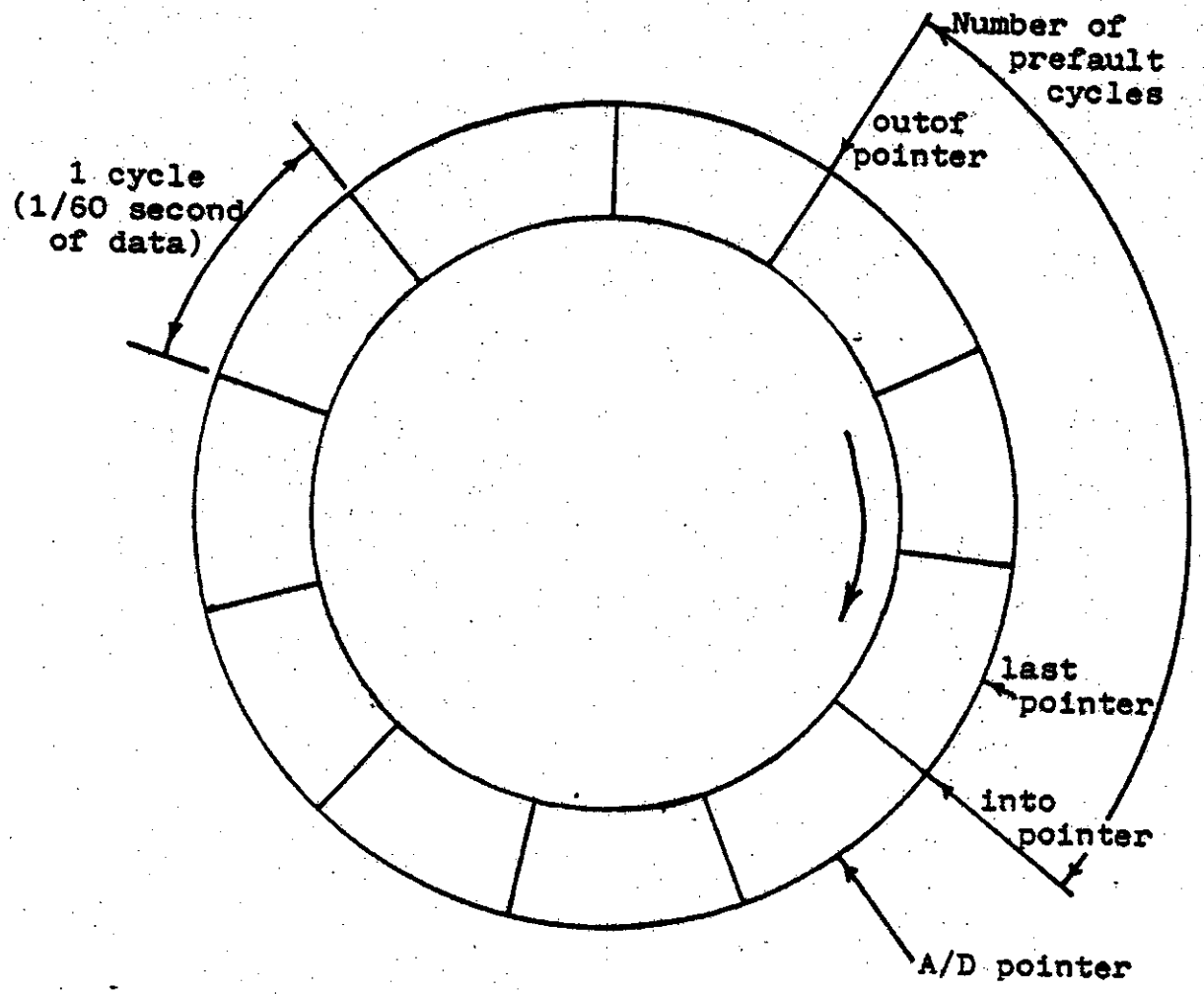


Figure 3.7 Acquired data ring buffer

changed state, no trigger could have occurred. Any bits that changed state are noted as having gone from the Ø to 1 state (leading edge), or from the 1 to Ø state (trailing edge). Any bits that went high are checked to see if such a transition is to cause a trigger, and if so, the trigger flag is set. The same operation is then performed on any bits that went low. The 'leading-edge', and 'trailing-edge'

masks are set from the event channel parameters.

The analog channels are then tested for a fault condition. A channel's current sample is compared with the sample on that channel taken one cycle (1/60'th second) earlier (lastpointer). If the change is greater than the user-specified limit for that channel, the trigger flag is set. Because of the time constraints on the monitor routine the event channels and the analog channels are not checked in the same invocation. In addition, only one quarter of the analog channels are checked each time. Thus all the event channels, and each analog channel are tested for a fault condition every fifth sample.

When all trigger testing has been completed the event data are packed in with the analog data as described in section 4.3. The monitor routine then exits if a full cycle (1/60'th second) has not yet been taken. If a full cycle has been taken (8 samples at 480 Hz sampling, or 16 at 960) the into pointer is updated to point at where the next cycle of information will be placed. If the updated into pointer is pointing to the same location as the outof pointer it means that the floppy disk has not been able to get the data out of memory fast enough. In this case a diagnostic message is printed and the program halts. If the into pointer has not overrun the outof pointer the program continues, testing the state of the recording bit in the system status word.

If this bit is off, the trigger flag is tested to see if a fault condition existed anywhere in the cycle. If not, the outof pointer is updated to point one cycle further along. If a fault was detected in the cycle the recording bit is set in the system status word and the postfault block counter is set to the user-specified number of postfault cycles to be recorded. The monitor routine then exits.

If the recording bit is on, the trigger flag is tested to see if a fault condition existed in the cycle. If not, the postfault counter is decremented. When this counter reaches zero the sampling bit in the system status word is cleared, and data acquisition for the fault terminates. If a fault was detected in the cycle the postfault counter is reset to the specified number of postfault cycles. The monitor routine then exits.

In summary, the into pointer is updated by the monitor routine every cycle. The outof pointer is updated by the monitor routine every cycle unless data are being recorded on the disk. In this case the disk controller software writes the data to which the outof pointer is pointing and then updates the outof pointer. The 'distance' by which the outof pointer lags the into pointer thus sets the prefault recording time. The postfault recording time specifies the amount of data to be recorded after the fault condition terminates. The number of cycles recorded for a particular fault is then:

$$N_{pre} + N_{post} + N_{fault} - 1$$

where N_{pre} is the specified number of prefault cycles to be recorded, N_{post} is the specified number of postfault cycles to be recorded, and N_{fault} is the number of cycles for which the fault condition exists.

3.5 Fault Data Manipulation

Writing fault data on the disk is the fundamental purpose of the recorder. This operation is handled by the start, and recordfault routines. The start routine is entered from the command decoder when the user enters the start command. It searches for the first free block on disk 1, removing any unused directory slots and empty data areas as it proceeds. If there is no free space on disk 1 an error message is printed and the command terminates. If free space does exist the start routine then sets up the trigger masks for the event channels, and the trigger limits for the analog channels. The a/d converter registers are initialized and the into and outof pointers are set up to reflect the desired prefault recording interval. The sampling bit in the system status word is set, the routine waits until the prefault recording interval has been satisfied, then a message is printed that sampling has started and the actual trigger parameters are set. The routine then enters a loop, waiting for the recording flag in the system status word to go high, or for an control-x to be typed on

the console. If the latter condition is met the sampling bit is cleared and the start command terminates. If the former condition is met the routine recordfault is called. When the recordfault routine returns the start routine loops, starting the process again at the point where free space is sought.

The recordfault routine first prints a message that recording is in progress. It then checks to see if the outof pointer has caught up to the into pointer. If the pointer has caught up the state of the sampling bit in the system status word is tested. If this bit is low it means the entire fault has been recorded, so a directory entry is written and the recordfault routine returns. If the sampling bit is high, the routine waits until the into pointer gets ahead of the outof pointer. The cycle to which the outof pointer points is then written on the disk and the outof pointer is updated. The routine then loops back to the point where the check for the outof pointer having caught up is made.

The other routines that manipulate the fault data are rmfault, rmlfault, and prfault. The rmfault and rmlfault routines remove all, or a portion of, data for a particular recording. To remove a fault entirely a -1 is placed in the last block entry of the directory. This directory slot is then noted as empty and will be overwritten when the start command is given. To remove portions of a fault the

first block and last block entries in the directory are modified to point to the first and last blocks respectively of the remaining data on disk 1. A special case of the rmfault routine (rm *) causes the entire directory space to be cleared. The prfault routine displays the data for a specified fault. This procedure is described in the following section.

3.6 Waveform Reconstruction and Display

It is not sufficient for the fault recorder simply to write the data on the disk and leave it at that. Some sort of display of the recorded data must also be provided. For the event channels this is quite simple; a table of event states is printed on the terminal with a line produced for each change of event state. The analog channels pose a somewhat larger problem because of the quantized nature of the recorded data. For sine waves about 25 samples per cycle are necessary for a dot display to give an accurate representation of the signal. Joining the data points with straight lines improves this to about 10 samples per cycle. Use of a waveform interpolation routine allows accurate display of sinusoidal signals sampled only 2.5 times per cycle. This allows the third harmonic of a 60 Hz signal, sampled at 480 Hz, to be displayed. The routine interpolate implements the algorithm which is described fully in appendix 6.2. The first and last cycles of a fault can not be displayed correctly because the algorithm uses data points

preceding and following the point being interpolated.

The routine graphchan displays the analog data, calling on interpolate to provide intersample analog values. The waveform may be displayed on the terminal in tabular form with or without an accompanying graph. The graph produced resembles the output of a strip chart recorder, but is poor quality because of the low resolution of the terminal. Alternately, the waveform may be displayed on a strip chart recorder connected to the fault recorder d/a outputs. In this case, the interpolated points are sent to the digital to analog converter at a rate of 30 points per second. The resultant time scaling is specified by the following equation.

$$\frac{\text{chart time}}{\text{real time}} = \frac{720}{\text{waveform display increment (degrees)}}$$

Thus if the waveform abscissa increment is set to 3 degrees (of a 60 Hz cycle) the time scaling will be 240:1. The time scaling allows low-cost, low frequency response, strip chart recorders to be used as a fault waveform display device. The fault recorder is not capable of driving several d/a display channels simultaneously.

3.7 Miscellaneous Routines

The other major routines in the fault recorder are lsl, which lists various parameters and ex, which examines the input channels. The ex routine is interesting in its treat-

ment of subcommand characters. Before calling the routine getchar to read a character from the console the routine raw is called. This routine sets the terminal raw i/o mode, which causes characters to be passed to the program as soon as they are typed. Thus the action for the subcommand characters is taken as soon as the character is typed; no carriage return is required. This treatment of the subcommand characters has two major side effects. The first is that since command action is taken as soon as the character is typed, there is no way of backspacing over, and reentering the command. The other is that all typeahead characters are thrown away when switching into and out of raw mode.

The routine main is the entry point for all C programs. The fault recorder main routine prompts the user for the sampling rate to be used, calls bootl to do some initialization, and then calls the routine shell. Shell reads a line from the console and breaks it into separate arguments. The first argument is taken to be a command name; the command table is searched, and if a match is found the routine corresponding to the matched name is called. The special routine Ø in the command table causes shell to exit. This is how the editor 'q' command operates. A prototype of the command arguments is also contained in the command table. The only purpose of this entry is to allow routines like help and puse to print the prototype argument in addition to the command name.

Menu driven operator interaction is provided by the nshell routine. The routine nshell prints the command names in the command table and then prompts the user to enter the number of the command to be executed. An invalid response causes the menu of commands to be redisplayed. A command address of Ø causes nshell to exit when that command is selected.

4. Evaluation of Recorder Performance

4.1 Introduction

After constructing and programming the fault recorder, an evaluation of its performance was made. The main concern was to demonstrate that the recorder could carry out the required functions. The results of tests performed in the laboratory are presented in the following section. The subsequent section describes some of the problems encountered after the recorder was installed for field tests. The final section of this chapter presents a brief analysis of the entire project, and gives some suggestions for further development of the recorder.

4.2 Laboratory Tests

The hardware configuration used for the recorder performance tests is shown in figure 4.1. A sweep frequency generator was used to simulate the power system analog signals, and a switch was used to simulate the event signals. The frequency generator was connected to analog input channel Ø (AØ) and the switch was connected to event input channel Ø (EØ). A permanent record of the test results was obtained by connecting a printer and strip chart to the unit.

Most of the test results shown in this section consist of a portion of the printer output and a corresponding strip

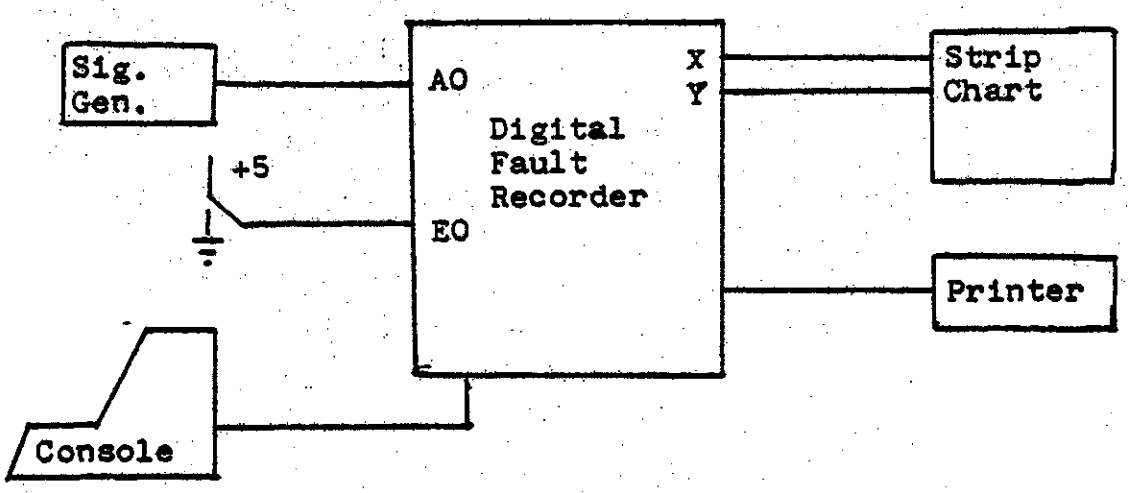


Figure 4.1 Laboratory test hardware connections

chart. Operator input is underlined to distinguish it from computer output. The strip charts are labeled with the chart speed in millimeters per second, and the time scaling in seconds of chart time to seconds of real time. The sensitivity of all the strip charts is 0.5 volts per millimeter.

The first test performed was a check on the power-up response of the recorder, and the handling of a few simple, commands. On power-up, the recorder first prompts for the sampling rate (480 or 960 Hz.) to be used. Once this parameter has been entered, the recorder then sizes its memory, prints out the ring buffer size, the maximum number of pre-fault cycles which may be recorded, and the default value of several system-wide parameters. The recorder then prints the 'fr>' (fault recorder) prompt and awaits operator input. This sequence is shown in figure 4.2. The prompt and

```
DFR-03 INITIALIZED

BUFFER SIZE 27904 BYTES. ROOM FOR 94 PREFAULT CYCLES.

RECORDING LABEL: ""
  4 PREFAULT CYCLES
 10 POSTFAULT CYCLES
480 Hz SAMPLING FREQUENCY
16 ANALOG CHANNELS
INTERNAL CLOCK
TABLE WITH GRAPH IN 12.00 DEGREE STEPS


FR>HELP
COMMANDS PRESENTLY IMPLEMENTED ARE:

ED [F#]
EX [AE]#
GLOBAL
HELP
LS -[AEFGT]
RM [*] [F#] [F#] ...
START
UTIL
```

Figure 4.2 Power-up Response

response for the sampling frequency (in this case 480 Hz) are not shown in the printout.

The signal generator was set to produce a 60 Hz, 2.4 volt rms sine wave. This signal was applied directly to analog input channel 0, bypassing the anti-aliasing filters. Recorder operation with this signal applied is shown in figure 4.3. This provides an example of the examine command. First the command is given, specifying analog channel 0 as the channel to be examined. The recorder then prints out the channel number, the channel name (blank in this case), and the channel triggering parameter. The ':' command is

```
FR>EX A0

A0                                        100.0% :"SIGNAL GENERATER<<OR"

A0  SIGNAL GENERATOR                      100.0% / 2.4V RMS
A0  SIGNAL GENERATOR                      100.0% =(.1 TO 100) 10

A0  SIGNAL GENERATOR                       10.0% D # CYCLES (.1 TO 1680) :
  TIME   SIGNAL
  (SEC)  (VOLTS)
 0.0000  -0.47
 0.0006  -1.17
 0.0011  -1.80
 0.0017  -2.33
 0.0022  -2.75
 0.0028  -3.05
 0.0033  -3.23
 0.0039  -3.30
 0.0044  -3.23
 0.0050  -3.03
 0.0056  -2.68
 0.0061  -2.18
 0.0067  -1.54
 0.0072  -0.80
 0.0078   0.00
 0.0083   0.78
 0.0089   1.51
 0.0094   2.15
 0.0100   2.67
 0.0106   3.05
 0.0111   3.30
 0.0117   3.40
 0.0122   3.36
 0.0128   3.17
 0.0133   2.84
 0.0139   2.40
 0.0144   1.86
 0.0150   1.27
 0.0156   0.62
 0.0161  -0.07
 0.0167  -0.78
A0  SIGNAL GENERATOR                       10.0%
```

Figure 4.3 Recorder Operation 60 Hz 2.4 vrms

then given to assign a name to the channel. The '<' character is printed by the fault recorder to indicate that the operator has erased a character by typing a backspace. The '/' command causes the fault recorder to print the rms value of the signal applied to the channel. A new triggering parameter is then entered with the '=' command. The recorder prints the valid range of response and the parameter is entered. Finally, a 'd' command is given to display the signal applied to the channel. Once again, the recorder prints the valid range of response, and in this case, a single cycle (at 60 Hz) of display is specified. A table and graph of the signal is then produced.

Setting up all the parameters (name and trigger) for all the analog and digital channels is obviously quite time consuming and requires a lot of typing. The fault recorder provides a means of saving the parameters on the floppy disk so they need not be reentered by hand each time the recorder is powered up.

The primary function of the fault recorder is, of course, to record faults. This operation is enabled by the 'start' command as shown in figure 4.4.

In this example a fault was simulated by turning off the signal generator. The "fault" was detected when samples from two subsequent cycles differed by more than 10%. The fault recorder recorded the signal waveforms, printed how much data was recorded, the time of the fault, and the

```
FR>START
SAMPLING STARTED
25 CYCLES RECORDED --- 49   8:48:33:7562
ROOM FOR 1977 MORE CYCLES
SAMPLING STARTED _
SAMPLING STOPPED
```

Figure 4.4 Start Command

amount of free disk space left. It then resumed monitoring for fault conditions. Finally, sampling was terminated by typing a 'control-X'. The fault waveform for channel Ø (signal generator) was displayed on the strip chart recorder as shown in figure 4.5.

The lower trace is the recorded signal. The upper trace is a marker which changes state at the point corresponding to the fault time printed by the recorder. The signal displayed before the marker goes high is the pre-fault portion of the recording. The fault recorder checks for the presence of a triggering condition every 1/6Ø'th of a second. This, and the relatively high analog triggering threshold (1Ø%), account for the delay between the time at which the signal begins to visibly decay and the time at which the fault is detected. The commands entered to produce this strip chart are shown in figure 4.6. The fault recorder prompts for the portion of fault data to be displayed, then indicates that the data are being displayed on the analog outputs. The time scaling is also printed.

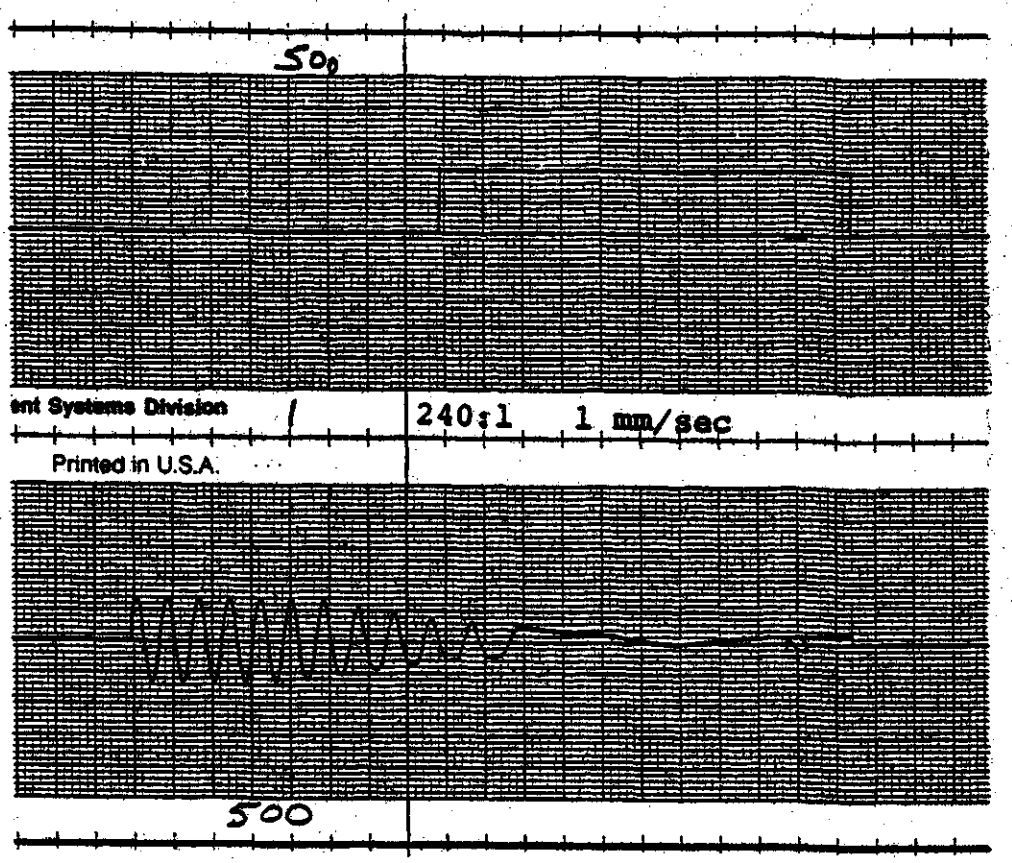


Figure 4.5 Fault 1 Channel Ø Signal

```
ED F0>DSP A0

A0  SIGNAL GENERATOR                          10.0%
FIRST CYCLE   NUMBER OF CYCLES TO DISPLAY : 1 23
WAVEFORM ON D/A (240.00 SECONDS PER SECOND)
ED F0>
```

Figure 4.6 Fault 1 Channel Ø Commands

In this example the 60 Hz input signal is displayed as a Ø.25 Hz signal.

The frequency response of the recorder was then tested by starting the recorder and sweeping the signal generator

from 60 to 180 Hz. The recorded data were then displayed on a strip chart as shown in figure 4.7. At the higher frequencies some distortion can be detected in the form of apparent amplitude modulation of the signal. The amplitude distortion is never larger than a few percent even at the highest (180 Hz) frequency recorded.

As an indication of the effectiveness of the waveform reconstruction algorithm the same signal was displayed without interpolating any points between the sampled values. As can be seen in figure 4.8 the display is unintelligible. Even at the lowest frequency shown (60 Hz) the sinusoidal nature of the waveform can not be determined.

The largest drawback of the reconstruction scheme is that samples both before and after the interpolation interval are used. Thus the first and last portions of a recording can not be interpolated properly. In order to provide at least some display of these regions, the fault recorder duplicates the first (or last) cycle of recorded data when the first (or last) cycle of a recording is displayed. This seems a reasonable thing to do, as presumably there are no large transients present at these points. This is not true under all triggering conditions however, so these regions must be regarded as approximate at best.

The fault recorder was then rebooted and the 960 Hz sampling rate was selected. This automatically reduces the number of analog channels to eight, as shown in the power-up

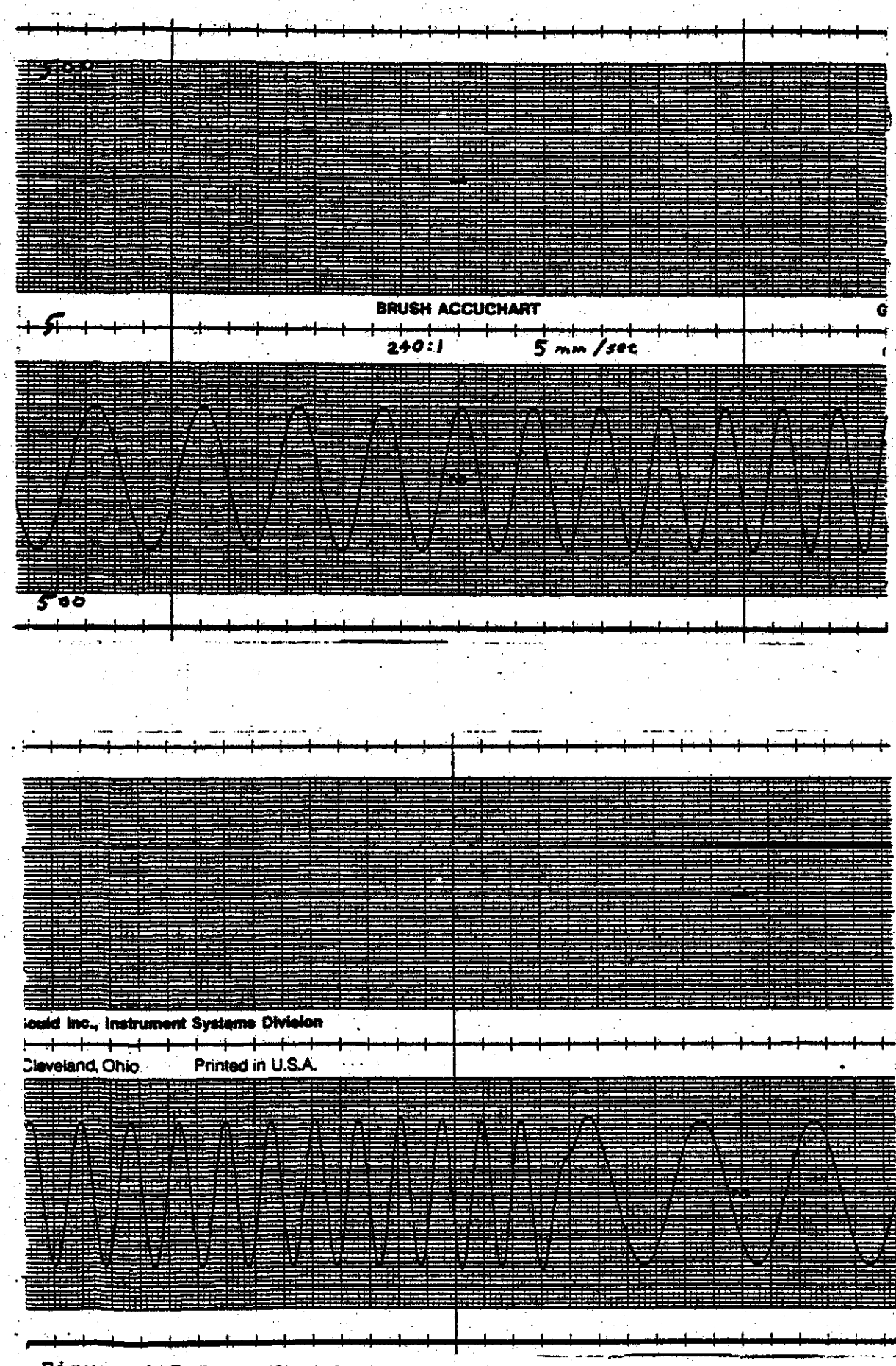


Figure 4.7 Recorder frequency response 60-180 Hz
480 Hz sampling

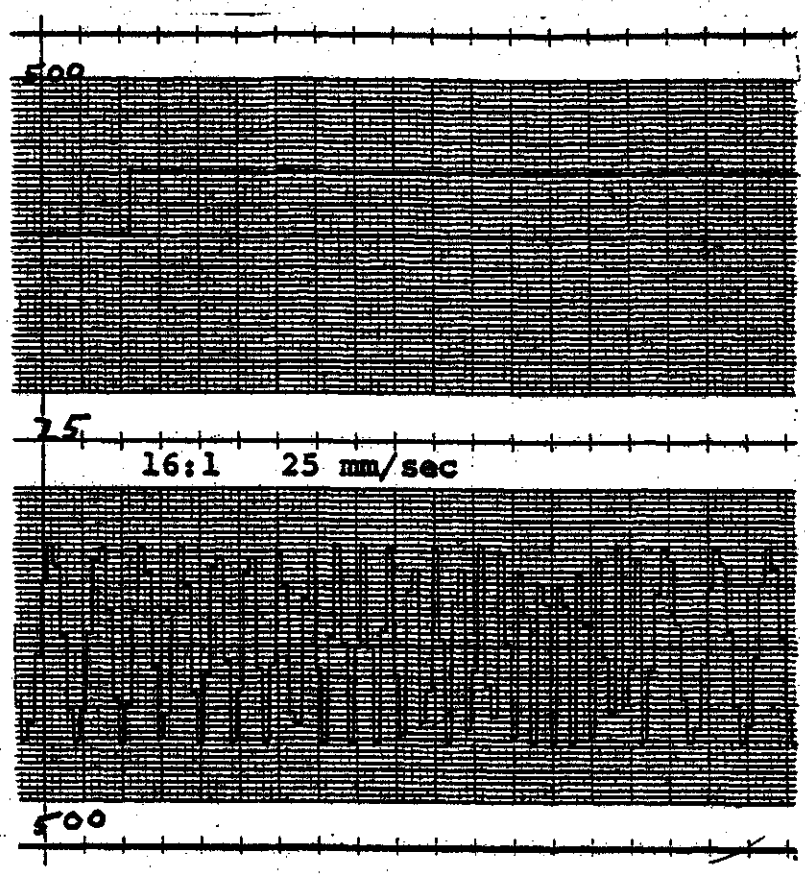


Figure 4.8 Recorder Frequency Response 60-180 Hz No waveform reconstruction

printout in figure 4.9.

The frequency response of the recorder was tested by sweeping the input signal from 60 to 350 Hz. The resultant recording is shown in figure 4.10. The 'roughness' of the display at the high frequencies is due to the limited number of points interpolated. Increasing the time scaling to 480 or 720 to 1 would produce a smoother display. The time to produce the display would, of course, be two or three times as long.

```
DFR-03 INITIALIZED

BUFFER SIZE 27904 BYTES. ROOM FOR 94 PREFAULT CYCLES.

RECORDING LABEL: ""
  4 PREFAULT CYCLES
 10 POSTFAULT CYCLES
960 HZ SAMPLING FREQUENCY
8 ANALOG CHANNELS
INTERNAL CLOCK
TABLE WITH GRAPH IN 12.00 DEGREE STEPS

FR>LS -A

A0                                  100.0%
A1                                  100.0%
A2                                  100.0%
A3                                  100.0%
A4                                  100.0%
A5                                  100.0%
A6                                  100.0%
A7                                  100.0%
```

Figure 4.9 960 Hz Sampling Power-up Printout

The event channels of the fault recorder were then given a rudimentary test. Event channel 0 was set to trigger on either the leading or trailing edge of its signal. Sampling was then started and the switch grounding event channel 0 was opened and closed. The display of the resultant fault data is shown in figure 4.11.

All the event channels are recorded and displayed even though only channel 0 was enabled for triggering. The first line of the display shows the state of the event channels at the beginning of the recording. Each time any event channel changes state a line is printed showing the time, and the

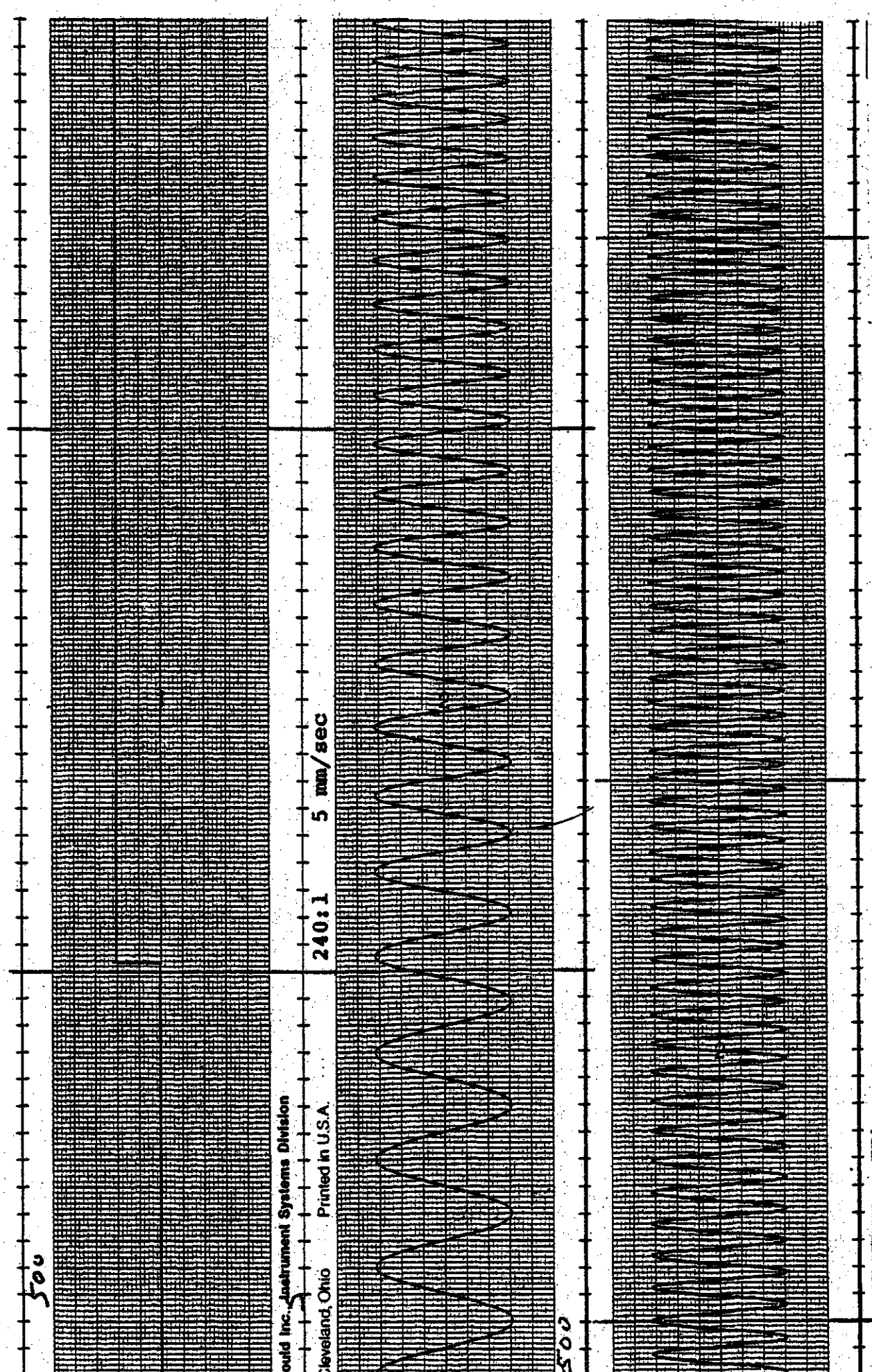


Figure 4.10 Frequency Response 60-350 Hz 960 Hz Sampling

```
FR>ED F0

 # CYCLES PRE POST RATE      TIME          CLOCK RECORDING LABEL
 0    58    4   10  960   0  0:15:12.8729 I   ""

ED F0>DSP E

     TIME          E15 E14 E13 E12 E11 E10  E9  E8    E2  E1  E0
 0  0:15:12:7906   1   1   1   1   1   1   1   1     1   1   0
 0  0:15:12:8656   1   1   1   1   1   1   1   1     1   1   1
 0  0:15:13:6052   1   1   1   1   1   1   1   1     1   1   0
```

Figure 4.11 Event Channel Test

new state of the event channels.

4.3 Field Test Results

During final testing of the fault recorder it was discovered that the memory board purchased for the recorder was slower than the memory board on which the system had been developed. With the slower memory installed the fault recorder would not operate at the 960 Hz sampling rate. The slow memory has recently been replaced and the recorder now operates at either sampling frequency.

Several problems surfaced when the fault recorder was installed in SPC's Condie switching station for field testing. The recorder would occasionally 'hang', not responding to fault conditions or to commands from the keyboard. The problem seemed to be caused by the analog to digital converter board; replacing the board eliminated the problem.

The second problem detected in the fault recorder was

related to the event channel triggering. The problem appeared when the recorder was set to trigger a recording on the low-to-high transition of any event channel. Under these conditions a recording would sometimes be triggered even when no fault condition (low-to-high transition) had occurred. The erroneous recording would continue until there was no space on the disk. The problem was traced to a bug in the system software. The software was patched on-site, and notations of the change were made in the program listing.

The third problem appeared after the recorder, with no external clock connected, had been left running for a few days. The day of year entry in the time and date printed by the system did not increment properly. The day would increment once (say from 196 to 197) but the next day would not (from 197 back to 196). This problem was also traced to a bug in the system software. Once again, the patch was minor enough that it could be done on-site.

A point of note is that both of the preceding bugs were in the portion of the software written in assembly language. No bugs have yet been discovered in the section of the software written in C.

The final problem was that, intermittently, the editor command would not work properly. This problem has not yet been fully rectified, but seems in some way to be related to electromagnetic noise affecting the system. After an

initial warmup period, and with the recorder shielded by its enclosure, the problem could not be made to occur. In any case, simply resetting the system cleared the problem. This problem was not felt to be too serious since it is intermittent, easily cleared, and does not affect the recording function of the system.

4.4 Conclusions and Recommendations

The following subsections review the original design specifications and describe the degree to which they were attained.

Intended Use

The main objective of this project was to develop a portable instrument capable of digitizing and recording samples of power-system voltage and current waveforms. The recorder has been in service for over one year and has recorded data from several system disturbances, thus indicating that the overall objective has been reached.

Sampling and Storage Capacity

The minimum objective of 240 Hz sampling on each of 7 analog and 16 digital (event) channels was surpassed by the recorder's ability to sample and record 8 analog and 16 digital channels at 960 Hz, or 16 analog and 16 digital channels at 480 Hz. The objective of recording 2 seconds of prefault and 120 seconds of postfault information was not

reached. The recorder is capable of recording only 1.8 seconds (109 cycles) of prefault and 31.5 seconds of postfault information. The computer's memory size placed the restriction on the prefault time and the disk capacity placed the restriction on the postfault time.

Triggering

The triggering objectives for the recorder were fully met. Recording may be triggered by any selected channel or channels. Triggering can occur when the rate of change of an analog signal exceeds a preset limit. Triggering is also possible from the change of state of a digital input (or inputs). The triggering parameters are easily set by commands from the operator's terminal.

Timing

The digital fault recorder incorporates all the timing capabilities as stated in the original design specification. The recorder incorporates a real-time-clock to set the sampling rate and to maintain the date and time. An external time signal (such as a satellite controlled clock) may be attached to the recorder. The optional satellite clock allows data from geographically remote recorders to be compared and analyzed on the same time base.

Accuracy and Range

The recorder meets the objectives of accuracy of 0.2%

of 1 p.u. for voltage signals and 1.0% of 1 p.u. for current signals. Voltage signals of 4 p.u. and current signals of 20 p.u. can be recorded without limiting thus meeting the objectives for signal range.

Interfacing Requirements

The Saskatchewan Power Corporation provided signal isolation and conditioning circuitry to convert the signals from standard PT's and CT's to the ±10 volt range accepted by the fault recorder. The recorder itself can not interface directly to the CT's and PT's.

Operator interaction with the recorder is through a standard hardcopy or video terminal as stated in the design objectives. The recorder was also to have been able to be accessed from a remote site through the telephone network. This should be simply a matter of replacing the operator's terminal with a modem connected to an ordinary telephone line, but has never been tested.

Packaging Requirements

The recorder was to be packaged in a portable enclosure and had to be capable of operating in an electrically noisy environment. Experience with the recorder has shown its ability to operate successfully in a switching station. The recorder may best be described as 'semi-portable' as it is quite large (about 1.2 meters high, 0.5 meters wide and 0.4 meters deep) and heavy (about 55 kg).

Power Calculation

The recorder objectives indicated the desirability of calculating the power flow in a transmission line in real-time. The computer used in the recorder was not fast enough to permit this calculation. This, and the postfault recording time, were the only major design objectives not attained.

Recommendations for Further Work

The fault recorder reflects the fact that it was designed for use as a research tool. Several changes would be necessary to make it a commercially acceptable product.

The recorder prompts and commands are quite terse. This is not a large problem now as the recorder is operated by Saskatchewan Power Corporation Research and Development personnel. A simpler, more 'friendly' form of interaction would be necessary if the recorder were to be run by Operations personnel.

The recorder can display only one analog channel at a time. A device capable of displaying several analog and event channels simultaneously would greatly increase the usefulness of the recorder. One possibility might be to use a video graphics terminal to display the data, with hardcopy available from an x/y or electrostatic plotter.

The sampling rate and number of channels is limited by

the rate at which data can be transferred to the disk. Replacing the floppy disks with a sealed, hard (Winchester) disk would make possible higher sampling rates and more channels. The best configuration would include both a Winchester disk and some sort of removable media device such as floppy disks or cartridge tapes. This would allow program and data transferral on the removable media and high speed recording on the fixed disk. A larger capacity disk would also increase the postfault recording time.

More local processing of the fault data should be provided. The recorder would be more useful if it could produce graphs of line impedances and power flows. Additional functions, such as load modelling might also be added to the recorder.

5. References

1. Carnahan, B., Luther, H. A., and Wilkes, J. O., Applied Numerical Methods, John Wiley and Sons, New York, New York (1969).

2. Carr, J., Brand, I. B., and Skelton, K. A., "Fault Data Recording, Event Recording and Time Base in Switching Station Automation", Report PE11-5550, Serial No. 6, SPC R & D Center (1975).

3. Carr, J. and Jackson, R. V., "Switching Station Automation Final Report", Report P311-5859, SPC R & D Center (1978).

4. Disturbance Monitoring by Telephone, Ontario Hydro Research Division Report, (July 9, 1976).

5. Hardy, S., Specifications for Electrical Transmission Disturbance Recording System, Design Proposal, March 14, 1978.

6. Kernighan, Brian W. and Ritchie, Dennis M., The C Programming Language, Prentice Hall, Englewood Cliffs, N.J. (1978).

7. Kernighan, Brian W. and Ritchie, Dennis M., "The UNIX Time-Sharing System", The Bell System Technical Journal, Vol 57 No 6 Part 2, pp 1905-1930, (July-August 1978).

8. Kernighan, Brian and Ritchie, Dennis, UNIX Programmer's Manual, Bell Telephone Laboratories, Murray Hill, N.J. (1975).

9. Linear Data Book, National Semiconductor Corporation, Santa Clara, California (1976).

10. Lycklama, H. and Christensen, C., "A Minicomputer Satellite Processor System", The Bell System Technical Journal, Vol 57 No 6 Part 2, pp 2103-2114 (July-August 1978).

11. Lycklama, H., "UNIX on a Microprocessor", The Bell System Technical Journal, Vol 57 No 6 Part 2, pp 2087-2102 (July-August 1978).

12. MCS-48 Microcomputer User's Manual, Intel Corporation, Santa Clara, California (1978).

13. MCS-85 Microcomputer User's Manual, Intel Corporation, Santa Clara, California (1978).

14. Memories and Peripherals, Microcomputer Handbook Series, Digital Equipment Corporation, Maynard, Mass. (1978).

15. Microcomputer Handbook, Second Edition, Digital Equipment Corporation, Maynard, Mass. (1977).

16. Model 1026 Satellite Controlled Clock Instruction Manual, Arbiter Systems Incorporated, Goleta, California (1977).

17. OS/8 Handbook, Digital Equipment Corporation, Maynard, Mass. (1974).

18. Osborne, A., An Introduction to Microcomputers Volume II, Some Real Products, Adam Osborne and Associates Inc., Berkeley, California (1977).

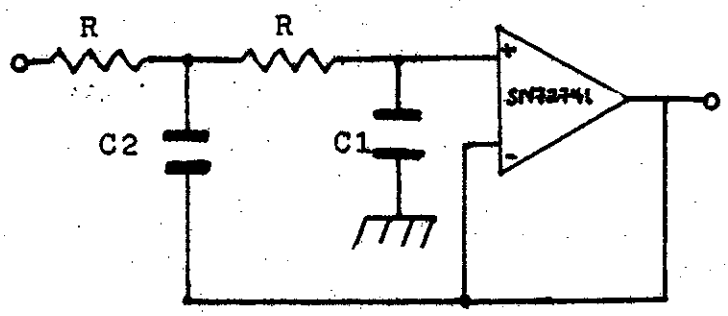
19. Ossana, J. F., Nroff User's Manual, Bell Laboratories, Murray Hill, N.J. (1974).

20. Sarkar, Surajit, Thesis Macros: A Tutorial, University of Saskatachewan, Saskatoon, Saskatchewan (1978).

6. APPENDICES

6.1 Low-Pass Filter Details

Anti-aliasing is provided by two banks of active second-order Butterworth filters. These filters are based on LM224 quad op-amps. One board has 16 filters with a cut-off frequency of 180 Hz and should be used when the recorder is set for 480 Hz sampling. The other board has filters with a 350 Hz cutoff and should be used when the recorder is set for 960 Hz sampling.



	R	C1	C2
180 Hz	34k	0.018uF	0.0039uF
350 Hz	95k	0.033uF	0.0068uF

The filter boards are connected to the fault recorder by a 50 pin ribbon cable. The pin description for this cable is included in the hardware description of the analog to digital converter board. The filters are connected to the front panel by a 40 pin ribbon cable. The pin description for this cable is:

1	NC	21	A/D 9
2	Digital Gnd	22	Analog Gnd
3	D/A X	23	A/D 8
4	Analog Gnd	24	Analog Gnd
5	D/A Y	25	A/D 7
6	Analog Gnd	26	Analog Gnd
7	NC	27	A/D 6
8	NC	28	Analog Gnd
9	A/D 15	29	A/D 5
10	Analog Gnd	30	Analog Gnd
11	A/D 14	31	A/D 4
12	Analog Gnd	32	Analog Gnd
13	A/D 13	33	A/D 3
14	Analog Gnd	34	Analog Gnd
15	A/D 12	35	A/D 2
16	Analog Gnd	36	Analog Gnd
17	A/D 11	37	A/D 1
18	Analog Gnd	38	Analog Gnd
19	A/D 10	39	A/D 0
20	Analog Gnd	40	Analog Gnd

LM124, LM224, LM324, LM2902

Specifications and Applications Information

QUAD LOW POWER OPERATIONAL AMPLIFIERS

The LM124 Series are low-cost, quad operational amplifiers with true differential inputs. These have several distinct advantages over standard operational amplifier types in single supply applications. The quad amplifier can operate at supply voltages as low as 3.0 Volts or as high as 32 Volts with quiescent currents about one fifth of those associated with the MC1741 (on a per amplifier basis). The common mode input range includes the negative supply, thereby eliminating the necessity for external biasing components in many applications. The output voltage range also includes the negative power supply voltage.

- Short Circuit Protected Outputs
- True Differential Input Stage
- Single Supply Operation: 3.0 to 32 Volts
- Low Input Bias Currents: 250 nA Max
- Four Amplifiers Per Package
- Internally Compensated
- Common Mode Range Extends to Negative Supply
- Industry Standard Pinouts

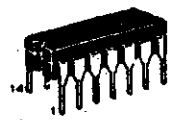
MAXIMUM RATINGS (T_A = +25°C unless otherwise noted)

Rating	Symbol	LM124 LM224 LM324	LM2902	Unit
Power Supply Voltages Single Supply Split Supplies	 V_{CC} V_{CC}, V_{EE}	 32 ±16	 26 ±13	Vdc
Input Differential Voltage Range (1)	V_{IDR}	±32	±26	Vdc
Input Common Mode Voltage Range (2)	V_{ICR}	-0.3 to 32	-0.3 to 26	Vdc
Input Forward Current (3) (V_I < -0.3 V)	I_{IF}	50	–	mA
Output Short Circuit Duration	t_S	Continuous		
Junction Temperature Ceramic and Metal Packages Plastic Package	T_J	 175 150		°C
Storage Temperature Range Ceramic and Metal Packages Plastic Package	T_{stg}	 -65 to +150 -55 to +125		°C
Operating Ambient Temperature Range LM124 LM224 LM324 LM2902	T_A	 -55 to +125 -25 to +85 0 to +70 –	 – – – -40 to +85	°C

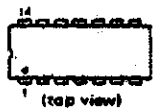
(1) Split Power Supplies.
(2) For Supply Voltages less than 32 V for the LM124/224/324 and 26 V for the LM2902, the absolute maximum input voltage is equal to the supply voltage.
(3) This input current will only exist when the voltage is negative at any of the input leads. Normal output states will reestablish when the input voltage returns to a voltage greater than -0.3 V.

QUAD DIFFERENTIAL INPUT OPERATIONAL AMPLIFIERS

SILICON MONOLITHIC INTEGRATED CIRCUIT



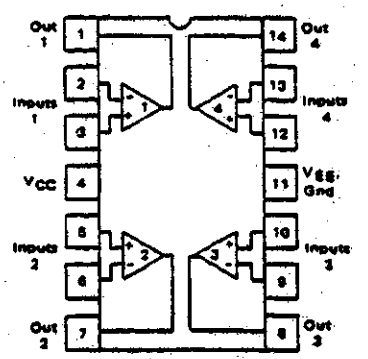
J SUFFIX
CERAMIC PACKAGE
CASE 632
TO-116



(top view)

N SUFFIX
PLASTIC PACKAGE
CASE 646
(LM224, LM324, LM2902 only)

PIN CONNECTIONS



(top view)

ORDERING INFORMATION

Device	Temperature Range	Package
LM124J	-55 to +125°C	Ceramic DIP
LM2902J	-40 to +85°C	Ceramic DIP
LM2902N	-40 to +85°C	Plastic DIP
LM224J	-25 to +85°C	Ceramic DIP
LM224N	-25 to +85°C	Plastic DIP
LM324J	0 to +70°C	Ceramic DIP
LM324N	0 to +70°C	Plastic DIP

ELECTRICAL CHARACTERISTICS (V_{CC} = 5.0 V, V_{EE} = Gnd, T_A = 25°C unless otherwise noted)

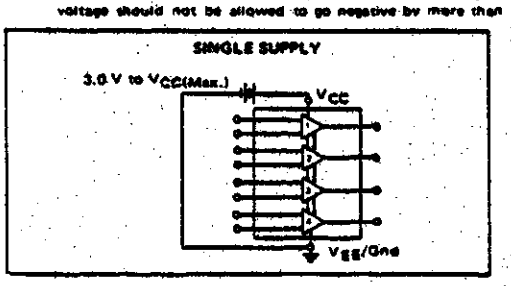
Characteristic	Symbol	LM124/LM224			LM324			LM2902			Unit
		Min	Typ	Max	Min	Typ	Max	Min	Typ	Max	
Input Offset Voltage V_{CC} = 5.0 V to 30 V (26 V for LM2902), V_{IC} = 0 V to V_{CC} − 1.7 V, V_O ≅ 1.4 V, R_S = 0 Ω	V_{IO}										mV
T_A = 25°C		–	2.0	5.0	–	2.0	7.0	–	2.0	7.0	
T_A = T_{high} to T_{low} (Note 1)		–	–	7.0	–	–	9.0	–	–	10	
Average Temperature Coefficient of Input Offset Voltage T_A = T_{high} to T_{low} (Note 1)	$\Delta V_{IO}/\Delta T$	–	7.0	–	–	7.0	–	–	7.0	–	μV/°C
Input Offset Current	I_{IO}	–	3.0	30	–	5.0	50	–	5.0	50	nA
T_A = T_{high} to T_{low} (Note 1)		–	–	100	–	–	150	–	45	200	
Average Temperature Coefficient of Input Offset Current T_A = T_{high} to T_{low} (Note 1)	$\Delta I_{IO}/\Delta T$	–	10	–	–	10	–	–	10	–	pA/°C
Input Bias Current	I_{IB}	–	-45	-150	–	-45	-250	–	-45	-250	nA
T_A = T_{high} to T_{low} (Note 1)		–	-50	-300	–	-50	-500	–	-50	-500	
Input Common-Mode Voltage Range (Note 2)	V_{ICR}										V
V_{CC} = 30 V (26 V for LM2902)		0	–	28.3	0	–	28.3	0	–	24.3	
V_{CC} = 30 V (26 V for LM2902), T_A = T_{high} to T_{low}		0	–	28	0	–	28	0	–	24	
Differential Input Voltage Range	V_{IDR}	–	–	V_{CC}	–	–	V_{CC}	–	–	V_{CC}	V
Large Signal Open-Loop Voltage Gain	A_{VOL}										V/mV
R_L = 2.0 kΩ, V_{CC} = 15 V, For Large V_O Swing, T_A = T_{high} to T_{low} (Note 1)		50 25	100 –	– –	25 15	100 –	– –	– –	100 –	– –	
Channel Separation 1.0 kHz ≤ f ≤ 20 kHz, Input Referenced	–	–	-120	–	–	-120	–	–	-120	–	dB
Common-Mode Rejection Ratio R_S ≤ 10 kΩ	CMRR	70	85	–	65	70	–	50	70	–	dB
Power Supply Rejection Ratio	PSRR	65	100	–	65	100	–	50	100	–	dB
Output Voltage Range R_L = 2 kΩ (R_L ≥ 10 kΩ for LM2902),	V_{OR}	0	–	3.3	0	–	3.3	0	–	3.3	V
Output Voltage−High Limit (T_A = T_{high} to T_{low})(Note 1)	V_{OH}										V
V_{CC} = 30 V (26 V for LM2902), R_L = 2 kΩ		26	–	–	26	–	–	22	–	–	
V_{CC} = 30 V (26 V for LM2902), R_L = 10 kΩ		27	28	–	27	28	–	23	24	–	
Output Voltage−Low Limit V_{CC} = 5.0 V, R_L = 10 kΩ, T_A = T_{high} to T_{low} (Note 1)	V_{OL}	–	5.0	20	–	5.0	20	–	5.0	100	mV
Output Source Current (V_{ID} = +1.0 V, V_{CC} = 15 V)	I_{O+}										mA
T_A = 25°C		20	40	–	20	40	–	20	40	–	
T_A = T_{high} to T_{low} (Note 1)		10	20	–	10	20	–	10	20	–	
Output Sink Current	I_{O-}										mA
V_{ID} = −1.0 V, V_{CC} = 15 V											
T_A = 25°C		10	20	–	10	20	–	10	20	–	
T_A = T_{high} to T_{low} (Note 1)		5	8	–	5	8	–	5	8	–	
V_{ID} = −1.0 V, V_O = 200 mV, T_A = 25°C		12	50	–	12	50	–	–	–	–	μA
Output Short Circuit to Ground (Note 3)	I_{OS}	–	40	60	–	40	60	–	40	60	mA
Power Supply Current (T_A = T_{high} to T_{low})(Note 1)	I_{CC}										mA
V_{CC} = 30 V (26 V for LM2902), V_O = 0 V, R_L = ∞		–	1.5	3.0	–	1.5	3.0	–	1.5	3.0	
V_{CC} = 5 V, V_O = 0 V, R_L = ∞		–	0.7	1.2	–	0.7	1.2	–	0.7	1.2	

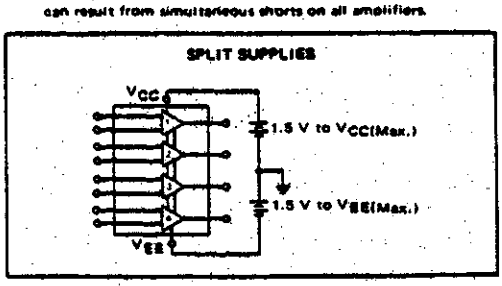
NOTES:

(1) T_{low} = −55°C for LM124, −40°C for LM2902, −25°C for LM224, 0°C for LM324; T_{high} = +125°C for LM124, +85°C for LM2902 and LM224, +70°C for LM324

(2) The input common-mode voltage or either input signal voltage should not be allowed to go negative by more than 0.3 V. The upper end of the common-mode voltage range is V_{CC} − 1.7 V, but either or both inputs can go to +32 V without damage (+ 26 V for LM2902).

(3) Short circuits from the output to V_{CC} can cause excessive heating and eventual destruction. Destructive dissipation can result from simultaneous shorts on all amplifiers.





6.2 Waveform Reconstruction Filter Details

The output $Y(j\omega)$ of a filter $H(j\omega)$ with input $X(j\omega)$ is given by

$$Y(j\omega) = H(j\omega) \cdot X(j\omega)$$

The time domain response $y(t)$ corresponding to $Y(j\omega)$ is given by the inverse Fourier transform:

$$y(t) = \frac{1}{2\pi} \int_{-\infty}^{\infty} Y(j\omega) e^{j\omega t} \, d\omega$$

If the input signal is an impulse (sample) with x_o amplitude then $X_o(j\omega)$ has an amplitude x_o for all frequencies. If this signal is passed through an ideal low-pass filter with cutoff frequency ω_c and gain $\frac{\pi}{\omega_c}$ in the passband, the output signal will be:

$$y_o(t) = \frac{x_o}{2\omega_c} \int_{-\omega_c}^{\omega_c} e^{j\omega t} \, d\omega$$

which reduces to the familiar sinc function:

$$y_o(t) = x_o \frac{\sin \omega_c t}{\omega_c t}$$

If the input signal is a series of impulses separated by the sampling interval T_s, where $T_s = \frac{1}{f_s}$ and the sampling frequency f_s is twice the filter cutoff frequency $f_s = \frac{\omega_c}{\pi}$, then the output of the filter will be the sum of all the $y_n(t)$.

$$y(t) = \sum_{n=-\infty}^{\infty} y_n(t)$$

$$= \sum_{n=-\infty}^{\infty} y(nT_s)$$

$$= \sum_{n=-\infty}^{\infty} x(nT_s) \frac{\sin(\omega_c(t-nT_s))}{\omega_c(t-nT_s)}$$

If τ is defined as the relative time $\tau = \frac{t}{T_s}$, then:

$$y(\tau) = \sum_{n=-\infty}^{\infty} x(n) \frac{\sin\pi(\tau - n)}{\pi(\tau - n)}$$

$$= \frac{\sin(\pi\tau)}{\pi} \sum_{n=\emptyset}^{\infty} (-1)^n \left[\frac{x(-n)}{\tau + n} + \frac{x(n+1)}{\sigma + n}\right]$$

where $\sigma = 1 - \tau$.

For reasons of practicallity, the sequence must be truncated at some point. The error introduced by this truncation may be minimized by applying suitable windowing coefficients:

$$y(\tau) = \frac{\sin(\pi\tau)}{\pi} \sum_{n=\emptyset}^{N} K_n (-1)^n \left[\frac{x(-n)}{\tau + n} + \frac{x(n+1)}{\sigma + n}\right]$$

The samples obtained from a sinusoidal signal of frequency ω and phase ϕ are given by:

$$x(n) = \sin\left(\frac{n2\pi\omega}{\omega_s} + \phi\right)$$

The maximum interpolation error will occur midway between two samples, where $\sigma = \tau = \frac{1}{2}$. At this point:

$$y = \frac{4}{\pi} \sin\left(\frac{\pi\omega}{\omega_s} + \phi\right) \sum_{n=\emptyset}^{N} K_n (-1)^n \frac{\cos\left(\frac{(2n+1)\pi\omega}{\omega_s}\right)}{2n+1}$$

At this point the value of the signal from which the samples were taken is given by:

$$x = \sin\left(\frac{\pi\omega}{\omega_s} + \phi\right)$$

Thus the relative error between x and y is:

$$e = \frac{X-Y}{X} = 1 - \frac{Y}{X}$$

$$= 1 - \frac{4}{\pi} \sum_{n=0}^{N} K_n (-1)^n \frac{\cos(\frac{(2n+1)\pi\mu}{\mu_s})}{2n+1}$$

For zero error:

$$\cos(\frac{\pi\mu}{\mu_s}) - \frac{\pi}{4} = \sum_{n=1}^{N} K_n (-1)^n \frac{\cos(\frac{(2n+1)\pi\mu}{\mu_s})}{2n+1}$$

Thus the error can be zeroed at N frequencies. The frequencies so chosen for the fault recorder were 0, 30, 60, 90, 120, 150 Hz for the 480 Hz sampling rate, and 0, 60, 120, 180, 240, 300, 345 Hz for the 960 Hz sampling rate. The resultant windowing coefficients are shown in the fault recorder listing.

6.3 Hardware Configuration

This appendix describes the hardware configuration of the fault recorder. See the hardware manuals for the particular boards for a description of how the switches/jumpers should be set to obtain the following parameters.

1. CPU Board
    a) Master Clock Enabled
    b) Bus Event Line Disabled
    c) Power-up Mode 2 (begin execution at 0173000)

2. Memory
    a) Start Address 0000000
    b) Size 30k Words

3. Serial Line Interface
    a) Base Address 0176500
    b) Base Vector 0300
    c) Channel 3 Console Device
        (address 0177560, vector 060)
    d) Reboot on Channel 3 Break
    e) Rates:
        Channel 0: not used, any rate acceptable
        Channel 1: Printer speed
        Channel 2: System Clock, 9600 baud
        Channel 3: Console terminal speed
    f) Data/parity/stop bit format (all channels)
        8 data bits
        1 stop bit
        No parity
    g) Serial line levels compatible with RS-232C

4. Analog I/O Board
    a) Address 0177000
    b) Vector 0130
    c) A/D +/-10 volt, Offset Biannary
    d) D/A +/-10 volt, Two's Complement

5. Floppy Disk
    a) Address 0177170
    b) Vector 0264
    c) Bootstrap Enabled
    d) Bootstrap Address 0173000
    e) RX02 mode

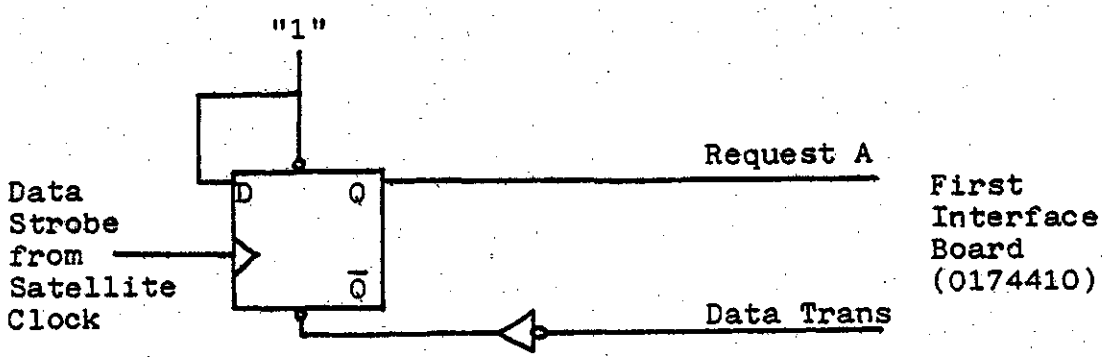
6. Event Inputs
    a) Address 0174400
    b) Vector 0150

7. Satellite Clock (Low Order)
    a) Address 0174410
    b) Vector 0160

8. Satellite Clock (High Order)
    a) Address 0174420
    b) Vector 0170

9. Terminator, Control
    a) KW11 Compatible

by a circuit as shown below.



This circuit latches the data strobe (pin 16) from the satellite clock, to provide a 'data ready' signal to the fault recorder. When the fault recorder reads the information, a narrow (750 nsec) pulse is generated on the DATA TRANS (J2-C) line which resets the strobe latch.

Note that the DT2768's do not latch the information on the data lines. The satellite clock should change the data on these lines only at the leading edge of the data strobe signal.

The full pinout for the DT2768's may be found in their hardware manuals.

6.5 Command Manual

The following three sections provide a concise description of the operation of the digital fault recorder.

The first section (I) describes the recorder level commands. The second section (II) describes the edit level commands. The third section (III) describes recorder typing conventions.

NAME

ed - edit fault data

SYNOPSIS

ed f#

DESCRIPTION

Ed switches the fault recorder from recorder mode to edit mode. The data recorded for fault f# may be displayed and portions removed. The prompt is changed from 'fr>' to 'ed f#>' (where f# is the argument given in the command) to indicate that the fault recorder is in edit mode.

The editor level subcommands are:

dsp	display analog or event data
global	change display parameters for this fault
help	print editor level commands
ls	list parameters for this fault
q	quit, go back to recorder mode
rm	remove portions of recorded information

The subcommands are described in detail in section 2 of this manual.

NAME

ex - examine an input channel

SYNOPSIS

ex [[ae]#]

DESCRIPTION

Ex prints the number, name, and triggering parameters for channel #. It then waits for a command character, which may be one of:

+ step to next highest channel.
Channel numbers 'wrap-around' so that stepping past the highest analog channel gets to the lowest event channel (e0). Stepping past the highest event channel gets to the lowest analog channel (a0).

- step to next lowest channel.
Stepping past the lowest analog channel (a0) gets to the highest event channel. Stepping past the lowest event channel (e0) gets to the highest analog channel.

/ for event channel, print current state (0 or 1).
for analog channel, sample a few cycles of channel #, compute the rms value of the signal (in volts), and print the result.

d Display channel
For analog channels the fault recorder prompts for the number of cycles to display. Channel # is then sampled and displayed in the current output format..
For event channels this command is identical to the '/' command.

= Set triggering parameters
For analog channels the '=' should be followed by a number indicating the trigger level for channel #. This number (taken as a percentage of the full scale +/-10 volt input range) specifies the change in values sampled at identical positions on successive cycles which will cause a recording. Setting this number to 100 effectively prevents the signal on channel # from causing a recording.
For event channels the '=' should be followed by two characters indicating the triggering parameters for the channel. The letter l and/or the letter t indicate that the leading (resp. trailing) edge of the signal on channel # is to cause a recording. Any other characters indicate that changes in the state of this channel are not to cause a recording.
Entering an invalid or no response after the equal command leaves the triggering parameter unchanged.
Note: All analog and event channels are recorded for the entire fault, regardless of their triggering parameters.

: Set channel name
The ':' should be followed by a channel name. The name must be enclosed in double quotes (") if it contains any spaces or tabs. Channel names may be up to 32 characters long.

\n (newline, or carriage return)
Terminate examine command. The fault recorder returns to recorder mode.

A list of the valid command characters will be printed if an illegal character is entered.

Typeahead does not work for the command character.

NAME

global - set global system parameters

SYNOPSIS

global

1. Change Recording Label
2. Set Prefault Recording Time
3. Set Postfault Recording Time
4. Set Waveform Display Format
5. Set Waveform Display Abscissa Increment
6. Terminate Command

Enter number of desired selection : #

DESCRIPTION

Global sets various system-wide (global) parameters. It prints a menu of subcommands, then prompts for a subcommand number. Entering an invalid command number, or no number at all, will cause the menu to be redisplayed. The subcommands are:

1. Change Recording Label
   The fault recorder will print the current recording label and then wait for a new recording label to be entered. The new recording label must be enclosed in double quotes (") if it contains any spaces or tabs. Entering only a carriage return to the prompt for a new recording label will leave the recording label unchanged.

2. Set Prefault Recording Time
   The fault recorder will print the current number of prefault cycles to be recorded and the valid range of the following input, and then wait for a number to be entered. Entering a number outside the specified range, or no number at all, will leave the prefault recording time unchanged.

3. Set Postfault Recording Time
   The fault recorder will print the current number of postfault cycles to be recorded and the valid range of the following input, and then wait for a number to be entered. Entering a number outside the specified range, or no number at all, will leave the postfault recording time unchanged.

4. Set Waveform Display Format
   The fault recorder asks if the analog waveforms are to be displayed on the terminal. If this question as answered affimatively the fault recorder will display the waveforms in a tabular form on the console terminal. The recorder then asks then asks if a graph as well as a table is desired. If the waveforms are not to be displayed on the terminal, they will be displayed by sending the values to

digital to analog converter 0 for ultimate display on a strip-chart recorder. The output rate is 30 points per second, so a 60 Hz signal displayed every 10 degrees will appear on output as a 0.83333 Hz signal. (72 seconds per second) A timing signal is present on d/a channel 1. This signal changes state at the point that corresponds to the fault time as displayed by the 'ls -f' command.

5. Set Waveform Display Abscissa Increment
The fault recorder prints the current abscissa increment and then prompts for a new abscissa increment (specified as a number of degrees of a 60 Hz signal). An invalid input, or no input at all, will leave the increment angle unchanged.

6. Terminate Command
The global command terminates.

NAME

help - list fault recorder commands

SYNOPSIS

help

DESCRIPTION

Help lists the fault recorder commands and their arguments. Optional arguments are shown enclosed in square brackets. A numeric argument is shown as a '#'. For example, the ls command is shown as :

```
ls    -[aefgt]
```

This means that the ls command may have an optional argument consisting of a dash (-) followed by any of the letters a, e, f, g, or t, or any combination of these letters.

NAME

ls - list information

SYNOPSIS

ls -[aefgt]

DESCRIPTION

Ls lists system information. The operation of this command is controlled by the argument, which must consist of a dash (-) followed by one or more of the following characters :

a List analog channel names and triggering parameters.

e List event channel names and triggering parameters.

f List faults recorded on disk. The number of cycles recorded, the number of pre and postfault cycles, the sampling rate, the time at which the fault occurred, the clock type, and the recording label are listed for each fault recorded.

g List global system parameters. The current recording label, the number of pre and post fault blocks to be recorded, the sampling frequency and clock type in use, and the analog waveform display parameters are listed. If the the stripchart form of analog waveform display is selected, the time base of the display is listed in terms of:
seconds of strip chart time / seconds of real time

t Print time and date.

If no argument is given '-gf' is defaulted.

NAME

rm - remove fault information

SYNOPSIS

rm [*] [f#] [f#] ...

DESCRIPTION

Rm removes the specified faults from the directory.

If the '*' argument is given, all faults will be removed from the directory.

In either case the user is asked if he is sure he wants to remove the faults.

The system keeps the fault numbers sequential by moving higher numbered faults down to the slots vacated by the removed faults. For example, assume that 10 faults have been recorded. They will be numbered f0 to f9. Further assume that faults f3 and f7 are not interesting, and are to be removed to free up some space on the disk. A 'rm f3 f7' command is given to remove the faults. If an 'ls -f' command is now given, the remaining faults will be numbered f0 to f7. i.e. f4 became f3, f5 became f4, f6 became f5, f8 became f6 and f9 became f7. The command to remove the two faults could have been given as two seperate commands, 'rm f7' and 'rm f3' (or 'rm f3' and 'rm f6', since for this case f7 has moved down to become f6). This lengthy example shows that care must be taken to insure that valuable fault data is not removed accidentally.

NAME

start - start sampling

SYNOPSIS

start

DESCRIPTION

Start initiates the recording operation of the fault recorder.

If any space on the disk has been freed by use of the 'rm' command the remaining data on the disk will be shuffled around to leave all the free space in one contiguous block. This operation is announced by the message 'Squishing data', or 'Squishing Directory' or both, and may take over a minute to complete. The control-x key is disabled so there is no way to abort the squishing.

The message 'Sampling Started' will be printed when the fault recorder has started taking data and checking for faults.

If a fault occurs, the message 'Recording' will be printed, followed by the time of the fault and the number of cycles recorded. Sampling will then be restarted if free space still remains on the disk.

The only command that is acknowledged during sampling is 'control-x' which stops sampling and returns to the command level. Any other characters entered will cause a question mark to be printed; sampling will continue unaffected.

util - utility command

SYNOPSIS

util

1. Store Parameters on Disk
2. Read Parameters from Disk
3. Set Time
4. Set Terminal Input/Output options
5. Format Disk
6. Copy Disk
7. Display Disk Statii
8. Reset System
9. Terminate Command

Enter number of desired selection : #

DESCRIPTION

Util is a 'catch-all' for several, little-used commands. It prints a menu of subcommands, then prompts for a subcommand number. Entering an invalid command number, or no number at all, will cause the menu to be redisplayed. The subcommands are:

1. Store Parameters on Disk
   Several global parameters are written onto the disk for subsequent retrieval. The parameters saved are:
   - The recording label.
   - The analog and event channel parameters, including channel names and triggering parameters.
   - The number of pre, and post-fault cycles to be recorded.
   - The analog waveform display format and abscissa increment.
   - The terminal input/output parameters.

2. Read Parameters from Disk
   The parameters saved on the disk by the above command are read from the disk. If the parameters on the disk are not valid, none of the fault recorder parameters will be changed.

3. Set Time
   The command requests the date and time in the following format;
   DDD HH MM [SS]
   Where DDD is the day of the year, HH is the hour, MM is the minute, and SS is the second. The seconds are optional and if not entered are set to 0. The command then prints the entered time and asks if it is correct. If the question is answered affirmatively the fault recorder's idea of the date and time is set. The date and time can not be set if an external satellite receiver is connected to the fault recorder.

4. Set Terminal Input/Output options

This command asks if the terminal is a paper (conversly video) terminal, if the output is to be sent to the printer port as well as the terminal, and if pause mode is to be selected. 'Paper-terminal mode' causes backspaces to echo as '<' characters. Normally they echo as <backspace>space<backspace>, which erases a character on a video terminal. Since the fault recorder has no printer, the printer mode should not be selected. 'Pause mode' allows the fault recorder to be used with terminals that use the XON/XOFF handshaking scheme. Upon receipt of an XOFF (control-s) character, the fault recorder will stop sending characters to the terminal until an XON (control-q) character is received.

5. Format Disk
The floppy disk in drive 1 is formatted in DEC double density format with a two sector interleave and a seven sector offset between tracks. This interleaving scheme is vital to the throughput of the disk system. All disks should be formatted before they are used in the fault recorder to prevent disk overrun errors caused by incorrect sector interleaving. All data stored on the disk are destroyed.

6. Copy Disk
The contents of the disk in drive number Ø are copied onto the disk in drive number 1. All the data previously stored on the disk in drive number 1 are destroyed.

7. Display Disk Statii
This command prints the contents of the RXØ2 floppy disk controller error registers and the status of each of the disk drives. The error register information is of use only after a disk error has occurred. Both drives must be 'Ready' and 'Hi Density' for the fault recorder to operate. If a drive is 'Ready', but 'Lo Density' it indicates that the disk in that drive needs to be formatted.

8. Reset System
A power-on-reset is simulated. The following operations are carried out :
   1. The system is rebooted from the floppy disk in drive Ø.
   2. A self-test on the hardware is carried out.
   3. All channel names, triggering parameters and global parameters assume their default values.
   4. An 'ls -g' command is executed to display the global parameters.
   5. The 'fr>' prompt is printed, and the command interpreter started.

9. Terminate Command
The util command terminates.

NAME

dsp - display fault data

SYNOPSIS

dsp [a#] [e]

DESCRIPTION

Dsp(II) is used to display fault data.

If the e argument is given, the event channels will be displayed in tabular form. The top line of the table will show the state of the event channels at the instant recording began. Each time an event channel changes state another line will be displayed showing the time and the state of all the event channels at that time.

If the a# argument is given, that analog channel will be displayed. The fault recorder will prompt for the first cycle to be displayed and the number of cycles to be displayed. Valid cycle numbers range from 0 to N-1, where N is the number of cycles recorded, as shown by the 'ls -f' command. These values will be prompted for until valid parameters are entered. The recorded signal will then be displayed in the current analog output format.

NAME

global - set global parameters

SYNOPSIS

global

DESCRIPTION

Global(II) is a subset of the global(I) command. The only global parameters that can be modifed are the analog output format, and the increment angle for the analog display.

The remaining global parameters are set from the data read from the disk directory for the fault when edit mode was entered.

NAME

help - list editor mode subcommands

SYNOPSIS

help

DESCRIPTION

Help lists the edit mode subcommands and their arguments. Argument notation is the same as help(I).

NAME

ls - list information

SYNOPSIS

ls -[aefgt]

DESCRIPTION

Ls(II) lists the information associated with the fault being edited. The operation of this command is controlled by the argument which must consist of a dash (-) followed by one or more of the following characters.

a List analog channel names, and triggering levels of the fault being edited.

e List event channel names and triggering parameters of the fault being edited.

f List the fault number, the number of cycles recorded, the number of pre and postfault cycles, the sampling rate, the time at which the fault occurred, and the recording label of the fault being edited.

g The analog waveform display parameters are listed. This command prints the current display parameters, not the parameters as they were when the fault occurred.

t Print time and date. This command prints the current time and date, not the time and date that the fault was recorded.

If no argument is given, '-gf' is defaulted.

NAME

q - quit

SYNOPSIS

q

DESCRIPTION

Q terminates the ed(I) command. This switches the fault recorder from edit mode back to recorder mode.

NAME

rm - remove portions of recorded data

SYNOPSIS

rm

DESCRIPTION

Rm removes all, or part of the data recorded for the fault being edited. The user is first asked if the entire fault is to be removed.

If an affirmative answer is given, the entire fault is deleted, and the fault recorder reverts to recorder mode. (since there is nothing left to edit for this fault)

If a negative answer is given, the user is asked how many prefault cycles he would like removed. The valid range of answers is printed, if the number entered is outside this range an error message is printed and the command aborted. The user is then asked how many postfault cycles he would like removed. Once again the valid range is printed; an invalid response terminating the command. If both responses were valid the appropriate number of cycles will be removed.

NAME

backspace - delete characters

SYNOPSIS

backspace key, or control-h

DESCRIPTION

Backspacing erases characters back to the beginning of a line. For example:

hh<backspace>ellp<backspace><backspace>p

is the same as "help", where <backspace> has been used to indicate the backspace character.

If the terminal i/o mode is set to 'Paper-Terminal mode', backspaces will echo as a left angle bracket '<'. Normally backspaces echo as <backspace>space<backspace>, which erases a character on a video terminal.

NAME

control x - abort command

SYNOPSIS

X^{c} - x key struck while cntl key is depressed

DESCRIPTION

Typing a control x causes the current command to be aborted.

Some critical commands cannot be aborted. For these commands the message 'Cannot abort command' is printed if a control x is typed. The command continues unaffected.

NAME

no - terminate parameter input

SYNOPSIS

no

DESCRIPTION

In most cases a response of no to a prompt for input will cause the command which is awaiting input to terminate. The only exception is for commands that are expecting a yes/no response.

For example, entering no to the util 'Enter number of desired selection : ' prompt, causes the util command to terminate.

NAME

carriage return/newline - enter line

SYNOPSIS

<return>, or <newline>, key

DESCRIPTION

All input to the digital fault recorder must be followed by a <return>. Striking this key signifies the end of the line of input.

Either the <return>, or <newline> key may be used, the effect is the same.

DESCRIPTION

The digital fault recorder has full typeahead, which means that characters may be entered as fast as desired, whenever desired, even when some command is typing. If typing is done during output, the input characters will appear intermixed with the output characters, but they will be stored away and interpreted in the correct order. Thus, commands can be entered one after another without waiting for the first to finish, or even begin.

NAME

yes/no - affirmative/negative answers

DESCRIPTION

All fault recorder prompts which expect a yes/no response look only at the first character of the response. If this character is 'y', or 'Y', the response is taken to be 'yes'. If the first character is 'n' or 'N', the response is taken to be 'no'. Any other character will cause the question to be asked again.

6.6 Recorder Program Listing

```
/*
 ************************************************************************
 *                                                                      *
 *              Parameter Definitions                                   *
 *                                                                      *
 ************************************************************************
 */


#define STACKSIZE 1024          /* allow 1k byte for stack */

#define NACHAN 16
#define NECHAN 16

#define NUNITS 2
#define LASTSECTOR 26
#define NTRACKS 77
#define SECTORSIZE 256          /* number of bytes in disk sector */

#define NOFAULT 010000          /* 100% change for analog channel */

#define RMSNUM 64               /* number of samples to take rms over */
#define RMSFULL 2684355.        /* rms value for 1 pu reading */
                                /* RMSFULL = (RMSNUM * ((2048**2) / 4) / 100) */
                                /* i.e. let full scale sine wave (-10v to 10v) */
                                /* be printed as 7.07 pu */
#define ANGLE2TIME 4.629629e-5  /* Conversion factor from waveform degrees to */
                                /* seconds. */
                                /* (1/60) / 360 */

#define DTHETAINC 12            /* default increment angle for table */

#define PI 3.141592654

#define ANALOG 0400             /* analog channel flag */

#define DPREFAULT 4             /* default number of prefault blocks (cycles) */
#define DPOSTFAULT 10           /* default number of postfault blocks (cycles) */
#define MAXPOSTFAULT 999        /* maximum post fault blocks */

/* special blocks on disks */
#define DTRACK 41
#define PTRACK 26
#define PARMSBLOCK PTRACK*26    /* parameters saved here */
#define DIRECTORY  DTRACK*26    /* start of directory */
#define NBLOCKS  77*26          /* last block on disk */
#define DBLOCKS (1+((1+sizeof parms)/SECTORSIZE))  /* number of disk blocks per direct

/* stty bits */
#define VIDEO 0100000
#define TABS 06000
#define TABS2 04000
#define RAW 040
#define CRMOD 020

/* data transfer commands */
```

```
#define B_READ 1
#define B_WRITE 2

/* status word bits */
#define SAMPLING 1
#define RECORDING 2

/* control x parameters */
#define INTERRUPT 070
#define CNTLX 030

/* serial line definitions */
#define DL11 0177560
#define DLIEN 0100

/* satellite clock definitions */
#define CLOCK 0174410

/* dlv11j clock definitions */
#define DLCLK 0176520
#define CLOCK_VECTOR 0324

/* floppy disk definitions */
#define RX02 0177170
#define RXERROR 0100000
#define INIT 040000
#define A17 020000
#define A16 010000
#define RX02_MODE 04000
#define HI_DENS 0400
#define TR_REQ 0200
#define RXIEN 0100
#define RXDONE 040
#define UNIT_1 020
#  define FILL_BUFF 00
#  define EMPTY_BUFF 02
#  define WRITE_SECTOR 04
#  define READ_SECTOR 06
#  define SET_DENSITY 010
#  define READ_STATUS 012
#  define WRITE_DEL_DATA 014
#  define READ_ERROR 016
#define RXGO 01
#define IBM_SD 0152
#define DEC_DD 0153
#define INIT_DONE 04


/* analog to digital converter definitions */
#define DT1761 0177000
#define DTDMA_VECTOR 0134
#define DTGO 01
#define DMA_EN 02
#define GS0 04
#define GS1 010
#define EXT_EN 020
```

```
#define RTC_EN 040
#define DTIEN 0100
#define DTREADY 0200
#define INC 0100
#define DTERROR 0200
#define DTMASK 07777
#define DTOFFSET 04000
#define DTRAWOFFSET 0174000
#define DTDA_X 0100000

/* Parallel line definitions */
#define DR11 0174400
```

```
/*
 ***************************************************************************
 *                                                                         *
 *                Macros                                                   *
 *                                                                         *
 ***************************************************************************
 */

#define max(a,b) ((a) > (b) ? (a) : (b))
#define min(a,b) ((a) < (b) ? (a) : (b))
```

```
/*
 ************************************************************************
 *                                                                      *
 *              Data Structure Definitions                              *
 *                                                                      *
 ************************************************************************
 */

/*
 * Structure to access an integer
 */
struct {
        int integ ;
        int integ2 ;
} ;

/*
 * Structure to access the words of a long integer
 */
struct {
        int hiword ;
        int loword ;
} ;

/*
 * The structure of the command table
 */
struct cmd {
        char    *cmd_name ;
        char    *cmd_options ;
        int     (*cmd_addr)() ;
} ;

/*
 * The structure of the time data
 */
struct tdata {
        char    b_sec ;
        char    b_min ;
        char    b_hr ;
        char    b_x0 ;
        int     b_day ;
        int     c_ticks ;
        char    c_sec ;
        char    c_min ;
        char    c_hr ;
        char    b_x1 ;
        int     c_day ;
} ;

/*
 * The structure of the raw time data
 */
struct tbuf {
        long    basetime ;
```

```
        long    offset ;
} ;

/*
 * The structure of the parameters of each analog channel
 */
struct aparm {
        char    name[32] ;
        float   delta_trig ;
} ;

/*
 * The structure of the parameters of each event channel
 */
struct eparm {
        char    name[32] ;
        char    lead_trig ;
        char    trail_trig ;
} ;

/*
 * The structure of a disk directory element
 */
struct direct {
        int     first_block ;           /* first disk block of fault */
        int     last_block ;            /* last disk block of fault */
                                        /* must be first 2 elements of struct */
        int     ext_clock ;             /* external or internal clock */
        int     nchan ;                 /* number of analog channels */
        int     smpcyc ;                /* number of samples per cycle */
        struct  tbuf fault_time ;       /* fault time */
        int     rate ;                  /* sampling rate */
                                        /* remaining variables are saved */
                                        /* and restored */
        char    dsname[32] ;            /* data set name */
        struct  eparm eparms[16] ;      /* event channel parameters */
        struct  aparm aparms[16] ;      /* analog channel parameters */
        int     pre_fault ;             /* number of prefault blocks */
        int     post_fault ;            /* number of postfault blocks */
        float   thetainc ;              /* table increment */
        int     tstate[3] ;             /* stty state */
        char    gflag ;                 /* flag for graph or just table */
        char    daflag ;                /* flag for table(graph) or analog */
                                        /* must be last item in structure */
                                        /* as daflag is used as terminator */
                                        /* for copying structures around */
} ;
#define BLK_NUM_SIZE (sizeof parms.first_block + sizeof parms.last_block)

/*
 * The structure of the rx02 error buffer
 */
struct rxebuf {
        char    errnum ;
        char    wcreg ;
        char    d0track ;
```

```
        char    dltrack ;
        char    ttrack ;
        char    tsector ;
        char    errstat ;
        char    track ;
} ;

/*
 * The structure of the linked list of pointers into adbuf
 */
struct dbuf {
        struct dbuf *b_forw ;
        int     *flt_data ;
} ;
```

```
/*
 ************************************************************************
 *                                                                      *
 *              Device Register Structures                              *
 *                                                                      *
 ************************************************************************
 */
/*
 * analog to digital converter
 */
struct dtl76l {
        char    dt_csrlo ;
        char    dt_csrhi ;
        int     dt_dbuf  ;
        int     dt_wcount ;
        int     dt_busadd ;
} ;

/*
 * satellite clock interface
 */
struct clock {
        int     c_csr ;
        int     c_xl ;
        int     c_rbufl ;
        int     c_x2 ;
        int     c_x3 ;
        int     c_x4 ;
        int     c_rbuf2 ;
} ;

/*
 * floppy disk
 */
struct rx02 {
        char rx_csrlo ;
        char rx_csrhi ;
        char rx_dblo ;
        char rx_dbhi ;
} ;

struct rx02_alt {
        int     rx_csr ;
        int     rx_db ;
} ;

/*
 * serial line
 */
struct dlll {
        char    dl_rcsr ;
        char    dl_x0 ;
        char    dl_rbuf ;
        char    dl_xl ;
        char    dl_xcsr ;
```

```
        char    dl_x2 ;
        char    dl_xbuf ;
        char    dl_x3 ;
} ;

/*
 * parallel line
 */
struct drll {
        int     dr_csr ;
        int     dr_xbuf ;
        int     dr_rbuf ;
} ;
```

```
/*
 ************************************************************************
 *                                                                      *
 *              External Variables                                      *
 *                                                                      *
 ************************************************************************
 */
#ifdef DEBUG
int debugf ;
#endif
int *adbuf ;                    /* buffer for a/d to dma into */
int buffersize ;                /* size of a/d buffer (words) */
int maxcycles ;                 /* # of cycles that fit in adbuf */
int maxdisplay ;                /* maximum number of cycles displayable */
int maxprefault ;               /* maximum number of prefault cycles */
int *adend ;                    /* marker for end of a/d buffer */

struct dbuf *dpntrs ;           /* linked list of pointers into adbuf */
struct dbuf *intopntr, *outofpntr ;     /* pointers to linked list */
int *lastpntr, *adpntr ;        /* pointers into adbuf */

int dtrig [NACHAN] ;            /* delta trigger limits */
int leadsig ;                   /* leading edge digital trigger */
int trailsig ;                  /* trailing edge digital trigger */

int reenter ;                   /* flag to detect overflow */
int sampno ;                    /* sample number << 12 */
int passno ;                    /* pass number (for triggering) */
int nbsamp ;                    /* number of bytes per sample */
int lastdig ;                   /* previous digital data */
int hnbsamp[4] ;                /* offsets into adbuf */
int allchan ;                   /* flag indicating all, or half channels */
int nchan ;                     /* number of channels */
int smpblk ;                    /* number of samples per disk block */
int adcounter ;                 /* another pass counter for fault detection */
int fltdet ;                    /* fault detected flag */
int maxachan ;                  /* maximum analog channel */
int maxechan ;                  /* maximum event channel */

int nrecord ;                   /* number of postfault blocks to write */
int nleft ;                     /* number of disk blocks left to write */
int nretry ;                    /* number of disk retries */

int status ;                    /* system status word */

struct direct parms ;           /* current parameters */
struct direct entry ;           /* directory of fault being examined */
struct rxebuf rxerr ;           /* error status buffer */

int rtticks ;                   /* counter for single channel analog i/o */
int rtfrq ;                     /* initial value of rtticks */
float nextpoint, lastpoint ;    /* limits for waveform interpolation */
int ipoint ;                    /* sample subscript corresponding to laspoint */
extern float k480[], k960[] ;   /* window  shaping coefficients ; */
```

```
int dblock ;                    /* block number of next directory entry */
int fblock ;                    /* block number of next fault entry */
int ftrack ;                    /* track number of next fault entry */
int fsector ;                   /* sector number of next fault entry */

int fnumber ;                   /* fault being examined */
int dblock ;                    /* directory block of fault being examined */
struct tbuf ctime ;             /* current time (raw) */
int ticks ;                     /* tick counter */
int iticks ;                    /* initial value of tick counter */

jmp_buf jbuf ;                  /* longjump environment */
```

```
#include <stdio.h>
#include <setjmp.h>
#include "c00.h"

/*
 * Initialization Routines
 */

/*
 * main -- entry point of program
 */
main()
{
        extern struct cmd cmdtable ;
#       ifdef DEBUG
        printf ("Entering main routine\n") ;
#       endif
        for (;;) {
                printf ("Enter sampling rate in Hz. (480 or 960) ") ;
                readf ("%d",&parms.rate) ;
                if ((parms.rate == 960) || (parms.rate == 480)) {
                        break ;
                }
        }
        chngps (PL4) ;
        bootl () ;
        setjmp (&jbuf) ;
        chngps (PL0) ;
        DT1761->dt_dbuf = 0 ;
        DT1761->dt_dbuf = 0 | DTDA_X ;
        while (shell ("\nfr>", &cmdtable, 0)) ;
#       ifdef DEBUG
        printf ("leaving main routine\n\n") ;
#       endif
}

/*
 * bootl -- test hardware, intitialize variables
 */
bootl()
{
        extern monitor(), rtcint() ;
        extern int argc ;
        extern char *argv[] ;
        extern reset() ;
        register struct aparm *apntr ;
        register struct eparm *epntr ;
        register int i ;
        char *cpntr ;
        extern char *_U_break ;
#       ifdef DEBUG
        extern char *_U_tsize ;
        extern char *_U_stacksize ;
#       endif

        if (parms.rate == 480) {
```

```
                nchan = 16 ;
                parms.smpcyc = 8 ;
                smpblk = 8 ;
                nbsamp = 32 ;
                allchan = 1 ;
                iticks = 2 ;
        }
        else {
                nchan = 8 ;
                parms.smpcyc = 16 ;
                smpblk = 16 ;
                nbsamp = 16 ;
                allchan = 0 ;
                iticks = 1 ;
        }
#       ifdef DEBUG
        if (debugf) {
                printf ("iticks %d\n", iticks) ;
        }
#       endif
        ticks = iticks ;
        parms.thetainc = DTHETAINC ;
        parms.pre_fault = DPREFAULT ;
        parms.post_fault = DPOSTFAULT ;
        parms.gflag = 1 ;
        parms.daflag = 0 ;
        for (i = 0 ; i < 4 ; i++) {
                hnbsamp[i] = nbsamp/4 * i ;
        }
        parms.nchan = nchan ;
        maxachan = nchan - 1 ;
        maxechan = NECHAN -1 ;
        if (CLOCK->c_csr >= 0) {
                parms.ext_clock++ ;
        }
        parms.dsname[0] = '\0' ;

        if ((RX02->rx_csr & RX02_MODE) == 0) {
                panic ("DISK NOT RX02") ;
        }
        DTDMA_VECTOR->integ = &monitor ;
        DTDMA_VECTOR->integ2 = PL4 ;
        attach (INTERRUPT, &reset, 0) ;
        CLOCK_VECTOR->integ = &rtcint ;
        CLOCK_VECTOR->integ2 = PL4 ;
        DLCLK->dl_xcsr = 101 ;
        DLCLK->dl_xbuf = 0 ;
        for (apntr = &parms.aparms[0] ; apntr < &parms.aparms[16] ; apntr++) {
                apntr->name[0] = '\0' ;
                apntr->delta_trig = 100 ;
        }
        for (epntr = &parms.eparms[0] ; epntr < &parms.eparms[16] ; epntr++) {
                epntr->name[0] = '\0' ;
                epntr->lead_trig = 0 ;
                epntr->trail_trig = 0 ;
        }
```

```
        i = (getsp () - sbrk (0)) - STACKSIZE ;
#       ifdef DEBUG
        if (debugf) {
                printf ("Break is at 0%o\nStack pointer 0%o\nAllocate size 0%o\n",
                        sbrk (0), getsp (), i) ;
        }
#       endif
        dpntrs = sbrk (i) ;
        maxcycles = i / (SECTORSIZE + sizeof *dpntrs) ;
        maxcycles = abs (maxcycles) ;
        buffersize = maxcycles * 128 ;
        adbuf = (dpntrs + maxcycles) ;
        adend = adbuf + buffersize ;
        if ((maxprefault = maxcycles - 15) <= 1) {
                panic ("Not Enough Memory") ;
        }
        maxdisplay = (maxcycles - 4) * 16 ;     /* must assume 16 samples per cycle */
                                                /* since may work with data of either rate */
                                                /* regardless of current clock rate */
        for (i = 0 ; i < maxcycles ; i++) {
                (dpntrs + i)->b_forw = (dpntrs + ((i+1)%(maxcycles))) ;
                (dpntrs + i)->flt_data = (adbuf + (i * 128)) ;
        }
#       ifdef DEBUG
        if (debugf) {
        printf ("dpntrs %o  adbuf %o  adend %o  maxcycles  %d buffersize %d\n",
                dpntrs, adbuf, adend, maxcycles, buffersize) ;
        printf ("maxdisplay %d\n", maxdisplay) ;
        }
#       endif
        status = 0 ;
        printf ("\nDFR-03 Initialized\n\n") ;
        printf ("Buffer size %u bytes. Room for %d prefault cycles.\n",
                buffersize * 2, maxprefault) ;
        argc = 2 ;
        argv[1] = "-g" ;
        lsl (&parms, -1) ;
}

/*
 * reset - execute non-local goto back to command level
 */
reset ()
{
        longjmp (&jbuf, 0) ;
}
```

```
#include <stdio.h>
#include <setjmp.h>
#include "c00.h"
/*
 ************************************************************************
 *                                                                      *
 *              Initialized External Variables                          *
 *                                                                      *
 ************************************************************************
 */
/*
 * Shell level commands
 */
struct cmd cmdtable [] {
# ifdef DEBUG
        "debug",noargs,&debug,          /* set/reset debugging flag */
        "dump",noargs,&dump,            /* dump out a disk block */
# endif
        "ed","[f#]",&edfault,           /* edit a fault */
        "ex","[ae]#",&ex,               /* examine a channel */
        "global",noargs,&glob,          /* set global system parameters */
        "help",noargs,&help,            /* print out current commands */
        "ls","-[aefgt]",&ls,            /* list parameters */
# ifdef DEBUG
        "quit",noargs,0,                /* exit (start transparent task */
# endif
        "rm","[*] [f#] [f#] ...",&rmfault,      /* remove faults */
        "start",noargs,&start,          /* start taking data */
        "util",noargs,&util,            /* utility routines */
        0
} ;

/*
 * The editor subcommands
 */
struct cmd edcmds [] {
# ifdef DEBUG
        "debug",noargs,&debug,          /* set/reset debugging flag */
        "dump",noargs,&dump,            /* dump out a disk block */
# endif
        "dsp","[a#] [e]",&prfault,      /* display fault info */
        "global",noargs,&exglob,        /* change display parameters */
        "help",noargs,&helpex,          /* print out current commands */
        "ls","-[aefgt]",&lsf,           /* list fault parameters */
        "q",noargs,0,                   /* go back to shell */
        "rm",noargs,&rmlfault,          /* remove a fault */
        0
} ;

/*
 * the utility command menu
 */
struct cmd utilmen[] {
        "Store Parameters on Disk",noargs,&save,
        "Read Parameters from Disk",noargs,&restore,
        "Set Time",noargs,&timeset,     /* set internal clock */
```

```
        "Set Terminal Input/Output options",noargs,&sttymode,
        "Format Disk",noargs,   &format,        /* format floppy disk */
        "Copy Disk",noargs,&rx_copy,
        "Display Disk Statii",noargs,&prxstat,  /* disk error registers */
        "Reset System",noargs,&rstsys,
        "Terminate Command",noargs, 0,
        0
} ;

/*
 * the global command menu
 */
struct cmd globmen[] {
        "Change Recording Label",noargs,&reclbl,
        "Set Prefault Recording Time",noargs,&preflt,
        "Set Postfault Recording Time",noargs,&postflt,
        "Set Waveform Display Format",noargs,&wavedsp,
        "Set Waveform Display Abscissa Increment",noargs,&absinc,
        "Terminate Command",noargs,0,
        0
} ;

/*
 * the window shaping coefficients
 */

float k480 [] {
        1.              ,
        -0.9900026      ,
        0.9273505       ,
        -0.7689065      ,
        0.5211294       ,
        -0.2548521      ,
        0.6549272e-01   ,
        0.
} ;

float k960 [] {
        1.              ,
        -0.9957045      ,
         0.9678518      ,
        -0.8924927      ,
         0.7589350      ,
        -0.5767922      ,
         0.3653717      ,
        -0.1315151      ,
        0.
} ;

/*
 * various messages
 */
char flthed[] {
"\n # cycles pre post rate      time       clock recording label\n" } ;

char noargs[] { "" } ;
```

```
char clktimeout[] {
"Clock Timeout" } ;

/*
 * Debugging flag
 */
#ifdef DEBUG
int debugf 1 ;
#endif
```

```
#include <stdio.h>
#include <setjmp.h>
#include "c00.h"

/*
 * utility functions
 */
/*
 * rx_wparm -- wait for rx02 transfer request flag, then send word
 */
rx_wparm (arg)
{
        register int timer ;
        timer = 20000 ;
        while (RX02->rx_csrlo >= 0) {
                if (--timer == 0) {
                        panic ("Disk Timeout") ;
                }
        }
        RX02->rx_db = arg ;
}

/*
 * rx_wcom -- write command to rx02
 *         -- return -1 if error set
 */
rx_wcom (arg)
{
        if (rx_chk() < 0) {
                return (-1) ;
        }
        RX02->rx_csr = arg ;
        return (0) ;
}

/*
 * rx_chk -- wait for disk to become ready
 */
rx_chk ()
{
        while ((RX02->rx_csr & RXDONE) == 0) ;

        if ((RX02->rx_csr < 0) || (RX02->rx_db & INIT_DONE)) {
#       ifdef DEBUG
        if (debugf) {
                prxerr () ;
        }
#       endif
                return (-1) ;
        }
        return (0) ;
}

/*
 * raw -- sets tty mode to raw
 */
```

```
raw ()
{
        int tbuf[3] ;
        gtty (0, tbuf) ;
        tbuf[2] =| RAW ;
        stty (0, tbuf) ;
}

/*
 * unraw -- sets tty mode to -raw
 */
unraw ()
{
        int tbuf[3] ;
        gtty (0, tbuf) ;
        tbuf[2] =& ~RAW ;
        stty (0, tbuf) ;
}

/*
 * rx_read -- read data from floppy disk
 */
rx_read (drive, block, buffer, nbytes)
{
        register int retry ;

        for (retry = 0 ; retry < 6 ; retry++) {
                if (rx_trans (B_READ, drive, block, buffer, nbytes) >= 0) {
                        return (0) ;
                }
                nretry++ ;
                printf ("retry\n") ;
        }
        prxerr () ;
        return (-1) ;
}

/*
 * rx_write -- write data to floppy disk
 */
rx_write (drive, block, buffer, nbytes)
{
        register int retry ;

        for (retry = 0 ; retry < 6 ; retry++) {
                if (rx_trans (B_WRITE, drive, block, buffer, nbytes) >= 0) {
                        return (0) ;
                }
                nretry++ ;
                printf ("retry\n") ;
        }
        prxerr () ;
        return (-1) ;
}

/*
```

```
 * rx_trans -- transfer data to/from disk
 *        -- arguments are command, drive number, block number,
 *              buffer address,  number of bytes to transfer (must be even)
 */
rx_trans (command, aunit, ablock, buf, nbytes)
 char *aunit ;
 char *ablock ;
 int *buf ;
 char *nbytes ;
{
        int track ;
        int sector ;
        register int ntrans ;
        register char *block ;
        register char *unit ;

        block = ablock ;
        unit = aunit ;

        if (unit >= NUNITS) {
                perr ("Bad Drive Number") ;
                return ;
        }
#       ifdef DEBUG
        if (debugf) {
        printf ("rx_trans: drive:%d track:%d sector:%d\n",unit,
                block/26,block%26+1) ;
        printf ("          buffer:%o nbytes:%d command:%d\n",buf,nbytes,
                command) ;
        }
#       endif
        unit =* 020 ;

        while (nbytes) {
                if (block >= NBLOCKS) {
                        perr ("Bad Block Number") ;
                        return (-1) ;
                }
                track = block / LASTSECTOR ;
                sector = (block % LASTSECTOR) + 1 ;
                ntrans = nbytes > SECTORSIZE ? SECTORSIZE : nbytes ;
                nbytes =- ntrans ;
                ntrans =/ 2 ;
                if (command == B_READ) {
                        if (rx_wcom (HI_DENS | unit | READ_SECTOR | RXGO) < 0) {
                                return (-1) ;
                        }
                        rx_wparm (sector) ;
                        rx_wparm (track) ;
                        if (rx_wcom (HI_DENS | EMPTY_BUFF | RXGO) < 0) {
                                return (-1) ;
                        }
                        rx_wparm (ntrans) ;
                        rx_wparm (buf) ;
                }
                else {
```

```
                        if (rx_wcom (HI_DENS | FILL_BUFF | RXGO) < 0) {
                                return (-1) ;
                        }
                        rx_wparm (ntrans) ;
                        rx_wparm (buf) ;
                        if (rx_wcom (HI_DENS | unit | WRITE_SECTOR | RXGO) < 0) {
                                return (-1) ;
                        }
                        rx_wparm (sector) ;
                        rx_wparm (track) ;
                }
                buf =+ ntrans ;
                block++ ;
        }
        return (rx_chk ()) ;
}

/*
 * prxerr -- print out current rx02 error buffer
 */
prxerr ()
{
        register int i ;

        printf ("RX02 Error Status\n") ;
        printf ("CSR %o\nBUF %o\n",RX02->rx_csr,RX02->rx_db) ;
        i = 30000 ;
        while ((RX02->rx_csr & RXDONE) == 0) {
                if (--i == 0) {
                        printf ("Disk Timeout\n") ;
                        return ;
                }
        }
        RX02->rx_csr = READ_ERROR | RXGO ;
        rx_wparm (&rxerr) ;
        rx_chk() ;
        printf ("Error Code %o\nWord Count %d\nDrive 0 Track %d\nDrive 1 Track %d\nTarg
          rxerr.errnum & 0377, rxerr.wcreg & 0377,
          rxerr.d0track,rxerr.d1track,rxerr.ttrack,
          rxerr.tsector) ;
        printf ("%d disk retries since power up\n",nretry) ;
        i = rxerr.errstat ;
        printf ("Unit %d Selected\nHead %sLoaded\nDrive 0 %s Density\nDrive 1 %s Densit
          i&0200?1:0,i&040?"":"Not ",i&020?"Hi":"Lo",i&0100?"Hi":"Lo",
          rxerr.track) ;
}

/*
 * prxstat -- print rx02 status
 */
prxstat ()
{
        register int i ;
        register int unit ;

        prxerr () ;
```

```
        printf ("CSR %o\n",RX02->rx_csr) ;
        for (unit = 0 ; unit < 2 ; unit++) {
                i = 30000 ;
                while ((RX02->rx_csr & RXDONE) == 0) {
                        if (--i == 0) {
                                printf ("Disk Timeout\n") ;
                                return ;
                        }
                }
                RX02->rx_csr = HI_DENS | unit<<4 | READ_STATUS | RXGO ;
                i = 30000 ;
                while ((RX02->rx_csr & RXDONE) == 0) {
                        if (--i == 0) {
                                printf ("Disk Timeout\n") ;
                                return ;
                        }
                }
                i = RX02->rx_db ;
                printf ("Unit %d ",unit) ;
                if (i & 0200) {
                        printf ("Ready, %s Density\n",i&040?"Hi":"Lo") ;
                }
                else {
                        printf ("Not Ready\n") ;
                }
        }
        if (RX02->rx_csr < 0) {
                prxerr () ;
        }
}

/*
 * getdata - read n samples from chan
 */
getdata (chan, n)
{
        register int *dpntr ;
        register int count ;
        register int i ;

#       ifdef DEBUG
        if (debugf) {
                printf ("getdata (0%o, %d)\n", chan, n) ;
        }
#       endif
        dpntr = adbuf ;
        count = n ;
        rtsetup (1) ;
        do {
                rtwait () ;
                DT1761->dt_csrhi = chan ;
                while (DT1761->dt_csrlo >= 0) ;
                *dpntr++ = DT1761->dt_dbuf - DTRAWOFFSET ;
        } while (--count) ;
#       ifdef DEBUG
        if (debugf) {
```

```
                dpntr = adbuf ;
                count = n ;
                do {
                        if ((count & 07) == 0) {
                                printf ("\n") ;
                        }
                        printf ("%6o   ", *dpntr++) ;
                } while (--count) ;
                printf ("\n") ;
        }
#       endif

        nextpoint = 360. / parms.smpcyc ;
        lastpoint = 0 ;
        ipoint = 15 ;
}

/*
 * rtsetup -- set up for "real-time" routines
 */
rtsetup (irtticks)
{
#       ifdef DEBUG
        if (debugf) {
                printf ("rtsetup (%d)\n",irtticks) ;
        }
#       endif
        rtfrq = irtticks ;
        rtticks = rtfrq + 10 ;
}

/*
 * rtwait -- wait for clock to time out
 */
rtwait ()
{
        register int i ;

#       ifdef DEBUG
        if (debugf) {
                DT1761->dt_dbuf = 02000 ;
        }
#       endif
        i = 30000 ;
        if (rtticks < 0) {
                panic ("Clock overrun") ;
        }
        while (rtticks) {
                if (--i == 0) {
                        panic ("Clock Timeout") ;
                }
        }
        rtticks = rtfrq ;
#       ifdef DEBUG
        if (debugf) {
                DT1761->dt_dbuf = 0 ;
```

```
        }
#       endif
}

/*
 * rx_panic -- got a disk error when one cannot be afforded
 */
rx_panic ()
{
        chngps (0200) ;
        prxerr () ;
        panic ("Disk Failure") ;
}

/*
 * noints -- the interrupt vector is attached here during critical
 *        -- disk operations that must not be aborted
 */
noints ()
{
        printf ("Cannot abort command\n") ;
}

/*
 * roundup -- return smallest integer greater than or equal to fval
 */
roundup (fval)
 float fval ;
{
        register int sign ;
        register int ival ;

        sign = 0 ;
        if (fval < 0) {
                fval = -fval ;
                sign++ ;
        }
        ival = fval ;
        if (ival != fval) {
                ival++ ;
        }
        if (sign) {
                ival = -ival ;
        }
        return (ival) ;
}

/*
 * rstsys -- reset system
 */
rstsys ()
{
        if (ask ("Are you sure you want to reset the system? ")) {
                exit () ;
        }
}
```

```
/*
 * Commands
 */

#include <stdio.h>
#include <setjmp.h>
#include "c00.h"

/*
 * help -- print out main level commands
 */
help ()
{
        extern struct cmd cmdtable ;
        helpl (&cmdtable) ;
}

/*
 * helpex -- print out editor sublevel commands
 */
helpex ()
{
        extern struct cmd edcmds ;
        helpl (&edcmds) ;
}

/*
 * helpl -- print out available commands
 */
helpl (commands)
 struct cmd *commands ;
{
        register struct cmd *cmdptr ;

        cmdptr = commands ;
        printf ("Commands presently implemented are:\n\n") ;
        do {
                printf ("%s %s\n",(*cmdptr).cmd_name,(*cmdptr).cmd_options) ;
        } while ((*++cmdptr).cmd_name) ;
}

timeset ()
{
        timel (0) ;
}

/*
 * timel -- if called with nonzero argument, print current idea of system time
          -- if called with 0 argument, ask for time, then set time
 */
timel (command)
{
        register char *cp ;
        register int i ;
        int day ;
        int hr ;
```

```
        int minut ;
        int sec ;
        struct tdata prtime ;

        if (command == 0) {
                if (parms.ext_clock) {
                        perr ("Can't set external clock") ;
                        return ;
                }
                for (;;) {
                        printf ("Enter Date and Time (DDD HH MM [SS]) : ") ;
                        if ((i = readf ("%d%d%d%d",&day, &hr, &minut, &sec)) == 3) {
                                sec = 0 ;
                        }
                        else if (i < 0) {
                                        return ;
                        }
                        else if (i != 4) {
                                continue ;
                        }
                        if (((cp = sec) > 60) || ((cp = minut) > 60)
                            || ((cp = hr) > 24) || ((cp = day) > 366)) {
                                        continue ;
                        }
                        printf ("%d %d:%02d:%02d\n", day, hr, minut, sec) ;
                        if (ask ("Correct")) {
                                break ;
                        }
                }
                chngps (PL4) ;
                ctime.offset = 0 ;
                ctime.basetime = day / 100 ;
                ctime.basetime =<< 4 ;
                day =% 100 ;
                i = day / 10 ;
                ctime.basetime =| i ;
                ctime.basetime =<< 4 ;
                day =% 10 ;
                ctime.basetime =| day ;
                ctime.basetime =<< 2 ;
                i = hr / 10 ;
                ctime.basetime =| i ;
                ctime.basetime =<< 4 ;
                i = hr % 10 ;
                ctime.basetime =| i ;
                ctime.basetime =<< 3 ;
                i = minut / 10 ;
                ctime.basetime =| i ;
                ctime.basetime =<< 4 ;
                i = minut % 10 ;
                ctime.basetime =| i ;
                ctime.basetime =<< 3 ;
                i = sec / 10 ;
                ctime.basetime =| i ;
                ctime.basetime =<< 4 ;
                i = sec % 10 ;
```

```
                ctime.basetime =| i ;
#       ifdef DEBUG
                printf ("%O\n", ctime.basetime) ;
#       endif
                chngps (PL0) ;
                return ;
        }
        chngps (PL4) ;
        unpack (&ctime, &prtime) ;
#       ifdef DEBUG
        printf ("Base Time 0%O %D      Offset 0%O %D\n", ctime.basetime,
           ctime.basetime, ctime.offset, ctime.offset) ;
#       endif
        chngps (PL0) ;
        printf ("Time last set at %3d  %2d:%02d:%02d\nCurrent time %3d  %2d:%02d:%02d.%
                prtime.b_day, prtime.b_hr, prtime.b_min, prtime.b_sec,
                prtime.c_day, prtime.c_hr, prtime.c_min, prtime.c_sec,
                i = (prtime.c_ticks * 10.416667)) ;
}

/*
 * format -- format a floppy disk
 */
format()
{
        extern reset () ;
        char sectors[26] ;
        register int tracks ;
        register int i ;
        register char *spntr ;
        extern noints () ;

        if (ask ("\nFormatting destroys all the data stored on the disk.\nDo you really
                return ;
        }
        printf("Insert disk into drive number 1, type <return> to continue:") ;
        if (getargs () < 0) {
                return ;
        }
        attach (INTERRUPT, &noints, 0) ;
        for (tracks = 0 ; tracks < 77 ; tracks++) {
                i = (tracks * 7) % 26 ;
                for (spntr = 1 ; spntr < 14 ; spntr++) {
                        sectors[i++%26] = spntr ;
                        sectors[i++%26] = spntr + 13 ;
                }
                if (rx_wcom (UNIT_1 | WRITE_SECTOR | R/GO)< 0) {
                        return ;
                }
                rx_wparm (DEC_DD) ;
                rx_wparm (tracks) ;
                spntr = &sectors ;
                i = 26 ;
                do {
                        rx_wparm (*spntr++) ;
                } while (--i) ;
```

```
        }
        rx_chk () ;
        attach (INTERRUPT, &reset, 0) ;
}

/*
 * Start command -- start taking data
 */
start ()
{
        extern int argc ;
        extern char *clktimeout ;
        int ddtrig[NACHAN] ;
        int dleadsig ;
        int dtrailsig ;
        register int i ;
        register int bit ;
        register struct eparm *epntr ;
        struct dbuf *sintopntr ;
        long timeout ;
        struct tdata prtime ;

   for (;;) {
        i = DL11->dl_rbuf ;
        DL11->dl_rcsr = 0 ;

        if (findfreeblock () < 0) {
                status =& ~SAMPLING ;
                printf ("Sampling stopped\n") ;
                i = DL11->dl_rbuf ;
                DL11->dl_rcsr = DLIEN ;
                return ;
        }
        for (i = 0 ; i < 16 ; i++) {
                dtrig[i] = NOFAULT ;
                ddtrig[i] = parms.aparms[i].delta_trig * 40.96 ;
        }
        leadsig = 0 ;
        trailsig = 0 ;
        dleadsig = 0 ;
        dtrailsig = 0 ;
        bit = 1 ;
        for (i = 0 ; i < 16 ; i++) {
                epntr = &parms.eparms[i] ;
                if (epntr->lead_trig) {
                        dleadsig =| bit ;
                }
                if (epntr->trail_trig) {
                        dtrailsig =| bit ;
                }
                bit =<< 1 ;
        }
        sampno = 0 ;
        passno = 010 ;
        adcounter = smpblk ;
        fltdet = 0 ;
```

```
        i = DT1761->dt_dbuf ;
        DT1761->dt_csrlo = 0 ;
        DT1761->dt_csrhi = 017 ;
        timeout = 100 ;
        while (DT1761->dt_csrlo >= 0) {
                if (--timeout == 0) {
                        panic ("A/D converter failure") ;
                }
        }
        i = DT1761->dt_dbuf ;
        DT1761->dt_csrhi = INC | 017 ;
        DT1761->dt_wcount = ~16 ;

        outofpntr = dpntrs ;
        sintopntr = intopntr = dpntrs + parms.pre_fault ;
        i = intopntr->flt_data ;
        DT1761->dt_busadd = i ;
        adpntr = i ;
        lastpntr = i - nbsamp * (parms.smpcyc + 1) ;
        nrecord = parms.post_fault ;
#       ifdef DEBUG
        if (debugf) {
        printf ("adpntr %o  laspntr %o  \n",adpntr,lastpntr) ;
        printf ("intopntr %o  outofpntr %o\n",intopntr,outofpntr) ;
        printf ("into address %o  outof addres %o\n",intopntr->flt_data, outofpntr->flt_
        }
#       endif

        status = SAMPLING ;
        timeout.hiword = 010 ;  /* fudge long consant */
        timeout.loword = 0 ;
        while (outofpntr != sintopntr) {
                if (--timeout == 0) {
                        panic (&clktimeout) ;
                }
        }
        printf ("Sampling started\n") ;

        for (i = 0 ; i <= maxachan ; i++) {
                dtrig[i] = ddtrig[i] ;
        }
        leadsig = dleadsig ;
        trailsig = dtrailsig ;

        for (;;) {
                if (status & RECORDING) {
                        if (recordfault () >= 0) {
                                break ;
                        }
                }
                else {
                        if (DL11->dl_rcsr < 0) {
                                if ((DL11->dl_rbuf & 0177) != CNTLX) {
                                        DL11->dl_xbuf = '?' ;
                                        continue ;
```

```
                                }
                        }
                        else {
                                continue ;
                        }
                }
                status =& ~SAMPLING ;
                printf ("Sampling stopped\n") ;
                i = DLll->dl_rbuf ;
                DLll->dl_rcsr = DLIEN ;
                return ;
        }
    }
}

/*
 * rx_copy -- copy disk in drive 0 to disk in drive 1
 */
rx_copy ()
{
        extern noints () ;
        extern reset () ;
        register int nblklft ;
        register int ntrans ;
        register int block ;

        printf ("Insert source disk into drive 0, destination disk into drive 1.\nType
        if (getargs () < 0) {
                printf ("Copy not done\n") ;
                return ;
        }
        attach (INTERRUPT, &noints, 0) ;
        nleft = NBLOCKS ;
        block = 0 ;
        while (nleft) {
                nblklft = min (nleft, maxcycle) ;
                nleft =- nblklft ;
                if ((rx_read (0, block, adbuf, nblklft * SECTORSIZE) < 0)
                   || (rx_write (1, block, adbuf, nblklft * SECTORSIZE) < 0)) {
                        return ;
                }
                block =+ nblklft ;
        }
        attach (INTERRUPT, &reset, 0) ;
}

/*
 * perr - print error message
 */

perr (message)
        char *message ;
{
        printf ("Error -- %s\n", message) ;
}
```

```
/*
 * utility commands
 */

#include <stdio.h>
#include <setjmp.h>
#include "c00.h"

util()
{
        extern struct cmd *utilmen ;

        nshell (&utilmen) ;
}
/*
 * examine command
 */

ex (arg, cmdpntr)
        struct cmd *cmdpntr ;
{
        extern int argc ;
        extern char *argv[] ;
        float rmsval () ;
        float fval ;
        int analog ;
        register int chan ;
        char c ;
        register int i ;
        char strng[35] ;
        register int ppntr ;
        int ival ;

        chan = 0 ;
        analog = ANALOG ;
        c = 'a' ;
        if (argc == 2) {
                c = *argv[1] ;
                argv[1][0] = '0' ;
                chan = atoi (argv[1]) ;
        }
        if ((argc > 2) || ((c != 'a') && (c != 'e'))) {
                puse (cmdpntr) ;
                return ;
        }
        if (c == 'e') {
                analog = 0 ;
        }
        for (;;) {
                for (i = 0 ; i < (sizeof strng / sizeof c) ; i++) {
                        strng[i] = '\0' ;
                }
                if (analog) {
                        if (chan < 0) {
                                chan = maxechan ;
                                analog = 0 ;
```

```
                }
                else if (chan > maxachan) {
                        chan = 0 ;
                        analog = 0 ;
                }
        }
        else {
                if (chan < 0) {
                        chan = maxachan ;
                        analog = ANALOG ;
                }
                else if (chan > maxechan) {
                        chan = 0 ;
                        analog = ANALOG ;
                }
        }
        if (analog) {
                ppntr = &parms.aparms[chan] ;
        }
        else {
                ppntr = &parms.eparms[chan] ;
        }
        pparms (analog, chan, &parms) ;
        raw () ;
        i = (getchar () & 0177) | analog ;
        unraw () ;
        switch (i) {

          default :
                printf ("  [+ - : = / d]") ;
                break ;

          case '\n' :
          case '\n'+ANALOG :
          case CNTLX :
          case CNTLX+ANALOG :
                return ;

          case '+' :
          case '+'+ANALOG :
                chan++ ;
                break ;

          case '-' :
          case '-'+ANALOG :
                chan-- ;
                break ;

          case '/' :
          case 'd' :
                printf (" %c",(DR11->dr_rbuf & (1<<chan))?'1':'0') ;
                break ;

          case '/'+ANALOG :
                printf (" %.1gv rms", rmsval (chan)) ;
                break ;
```

```
                case 'd'+ANALOG :
                        printf ("  # cycles (.1 to %d) :", maxdisplay) ;
                        if (readf ("%f",&fval) <= 0) {
                                break ;
                        }
                        if ((fval > maxdisplay) || (fval < .1)) {
                                break ;
                        }
                        getdata (chan, ival = fval * parms.smpcyc + 60) ;
                        fval =* 360. ;
                        graphchan (0., fval, parms.smpcyc) ;
                        break ;

                case '=' :
                        if (readf ("%2s",strng) > 0) {
                                ppntr->lead_trig = 0 ;
                                ppntr->trail_trig = 0 ;
                                for (i = 0 ; i < 2 ; i++) {
                                        switch (strng[i]) {
                                          case 'l' :
                                                ppntr->lead_trig = 1 ;
                                                break ;
                                          case 't' :
                                                ppntr->trail_trig = 1 ;
                                                break ;
                                        }
                                }
                        }
                        break ;

                case '='+ANALOG :
                        printf ("(.1 to 100) ") ;
                        if (readf ("%f",&fval) <= 0) {
                                break ;
                        }
                        if ((fval >= 0.1) && (fval <= 100)) {
                                ppntr->delta_trig = fval ;
                        }
                        break ;

                case ':' :
                case ':'+ANALOG :
                        if (readf ("%32s", strng) >= 0) {
                                strncpy (ppntr->name, strng, 32) ;
                        }
                        break ;
                }
        }
}

/*
 * pparms - print parameters
 */
pparms (analog, chan, appntr)
{
```

```
        register int ppntr ;

        ppntr = appntr ;

        if (analog) {
                ppntr = &ppntr->aparms[chan] ;
        }
        else {
                ppntr = &ppntr->eparms[chan] ;
        }
        printf ("\n%c%-2d ",analog ? 'a' : 'e' , chan) ;
        printf ("%-32.32s ",ppntr->name) ;
        if (analog) {
                printf ("%5.1f%% ",ppntr->delta_trig) ;
        }
        else {
                printf ("   %c%c  ",ppntr->lead_trig ? 'l' : '-' ,
                        ppntr->trail_trig ? 't' : '-' ) ;
        }
}

/*
 * rmsval - take data from channel and return rms value
 *        - in fraction of full scale
 */

 float
rmsval (chan)
{
        register int count ;
        register int *dpntr ;
        float sqrt () ;
        float fsum ;
        float fval ;

        getdata (chan, RMSNUM) ;

        fsum = 0 ;
        count = RMSNUM ;
        dpntr = adbuf ;
        do {
                fval = *dpntr++ ;
                fsum =+ fval * fval ;
        } while (--count) ;

        return (sqrt (fsum / RMSFULL)) ;
}

/*
 * graphchan - print graph of channel values, every parms.thetainc degrees
 *             from theta = start to theta = limit
 */
graphchan (start, limit, scyc)
 float start ;
 float limit ;
{
```

```
        register int i ;
        register int j ;
        register int fillchar ;
        float interpolate () ;
        float fabs () ;
        float theta ;
        float volts ;

#       ifdef DEBUG
        if (debugf) {
                i = (limit * scyc) / 360. ;
                j = 0 ;
                printf ("%d samples\n", i) ;
                do {
                        if ((j & 07) == 0) {
                                printf ("\n") ;
                        }
                        printf ("%6.2f  ", adbuf[j++] / 204.8) ;
                } while (j < i) ;
                printf ("\n") ;
        }
#       endif
        if (parms.daflag == 0) {
                printf ("  Time  Signal\n  (sec) (volts)") ;
                fillchar = '-' ;
        }
        else {
                printf ("waveform on d/a ") ;
                darate () ;
                DT1761->dt_dbuf = 0 ;
                DT1761->dt_dbuf = 0 | DTDA_X ;
                rtsetup (parms.smpcyc * 2) ;
        }
        for (theta = start ; theta <= limit ; theta =+ parms.thetainc) {
                i = volts = interpolate (theta, scyc) ;
                if (parms.daflag) {
                        i = min (i, 03777) ;
                        i = max (i, -04000) ;
                        rtwait () ;
                        DT1761->dt_dbuf = i & DTMASK ;
                }
                else {
                        printf ("\n") ;
                        printf ("%7.4f %6.2f",theta*ANGLE2TIME,volts/204.8) ;
                        if (parms.gflag == 0) {
                                continue ;
                        }
                        i =/ 64 ;
                }
                if (fabs (theta) < parms.thetainc) {
                        if (parms.daflag) {
                                DT1761->dt_dbuf = 02000 | DTDA_X ;
                        }
                        else {
                                fillchar = '-' ;
                        }
```

```
                }
                if (parms.daflag) {
                        continue ;
                }
                for (j = -32 ; j <= 32 ; j++) {
                        if (j == i) {
                                printf ("*") ;
                                continue ;
                        }
                        if ((fillchar != ' ') && ((j % 8) == 0)) {
                                printf ("+") ;
                                continue ;
                        }
                        if (j == 0) {
                                printf ("|") ;
                                continue ;
                        }
                        if ((j < 0) || (j < i)
                                || ((j > i) && (fillchar != ' '))) {
                                printf ("%c",fillchar) ;
                                continue ;
                        }
                }
                fillchar = ' ' ;
        }
        if (parms.daflag) {
                DT1761->dt_dbuf = 0 ;
                DT1761->dt_dbuf = 0 | DTDA_X ;
        }
}

/*
 * interpolate -- return y value between samples
 */
 float
interpolate (angle, scyc)
 float angle ;
{
        float sum, sigma, tau ;
        float sin () ;
        register int i ;
        register float *kpntr ;
        float incpoint ;

#       ifdef DEBUG
        if (debugf) {
          printf ("angle %.7e   nextpoint %.7e   lastpoint %.7e\n",angle,nextpoint,last
        }
#       endif
        incpoint = 360. / scyc ;
        while (angle > nextpoint) {
                nextpoint =+ incpoint ;
                lastpoint =+ incpoint ;
                ipoint++ ;
        }
        while (angle < lastpoint) {
```

```
                lastpoint =- incpoint ;
                nextpoint =- incpoint ;
                ipoint-- ;
        }
        if (angle == nextpoint) {
                return (adbuf[ipoint+1]) ;
        }
        if (angle == lastpoint) {
                return (adbuf[ipoint]) ;
        }
        tau = (angle - lastpoint) / incpoint ;
        sigma = 1. - tau ;
        sum = 0 ;
#       ifdef DEBUG
        if (debugf) {
          printf ("sigma %.7e   tau %.7e\n",sigma, tau) ;
        }
#       endif
        kpntr = scyc == 8 ? &k480 : &k960 ;
        i = 0 ;
        while (*kpntr) {
                sum =+ *kpntr++ * ((adbuf[ipoint-i]/(i+tau)) +
                                   (adbuf[ipoint+1+i]/(i+sigma))) ;
                i++ ;
        }
        return (( sum * sin ( PI * tau )) / PI) ;
}
```

```
#include <stdio.h>
#include <setjmp.h>
#include "c00.h"

/*
 * sttymode -- select terminal type, output ports
 */
sttymode ()
{
        extern char palt ;
        register int mask ;
        register int i ;
        int state[3] ;

        mask = 0 ;
        gtty (0, state) ;
        i = state[2] ;
        printf ("Current mode is: %spaper ", i&VIDEO? "-": "") ;
        printf ("%sprinter ", palt ? "": "-") ;
        printf ("%spause\nNew mode:\n", (i&TABS)==TABS2? "": "-") ;
        if (ask ("video terminal")) {
                mask =| VIDEO ;
        }
        palt = ask ("printer on") ;
        if (ask ("pause mode")) {
                mask =| TABS2 ;
        }
        state[2] = (i & ~(TABS|VIDEO)) | mask ;
        stty (0, state) ;
}

/*
 * save -- save current parameters
 */
save ()
{
        gtty (0, parms.tstate) ;
        if (rx_write (0, PARMSBLOCK, &parms, sizeof parms) >= 0) {
                printf ("Parameters saved on disk\n") ;
        }
}

/*
 * restore -- restore current parameters
 */
restore ()
{
        register int i ;
        register int *ipntr ;
        register int *opntr ;

        if (rx_read (0, PARMSBLOCK, adbuf, sizeof parms) < 0) {
                return ;
        }
        i = ipntr = adbuf ;
```

```
        opntr = &parms ;
        if ((i->pre_fault < 1) || (i->pre_fault > 200) || (i->post_fault < 1)
          || (i->post_fault > 2000) || (i->thetainc < 0.1) || (i->thetainc > 60)) {
                perr ("Bad data on disk. Parameters not changed") ;
                return ;
        }
        opntr = &opntr->dsname[0] ;
        ipntr = &ipntr->dsname[0] ;
        while (opntr <= &parms.daflag) {
                *opntr++ = *ipntr++ ;
        }
        stty (0, parms.tstate) ;
        printf ("Parameters restored from disk\n") ;
}

/*
 * glob -- top level global command
 */
glob ()
{
        extern struct cmd globmen[] ;
        nshell (&globmen) ;
}

/*
 * exglob -- examine level global command
 */
exglob ()
{
        extern struct cmd globmen[] ;

        nshell (&globmen[3]) ;
}

/*
 * reclbl -- set recording label
 */
reclbl ()
{
        char string[40] ;

        printf ("Current recording label: \"%s\"\nEnter new recording label: ",
                parms.dsname) ;
        if (readf ("%32s",string) >= 0) {
                strncpy (parms.dsname, string, 32) ;
        }
}
/*
 * preflt -- set prefault recording time
 */
preflt ()
{
        int ival ;

        printf ("Current prefault recording time %d cycles\nEnter new prefault recordinc
        if((readf("%d",&ival)>0)&&(ival > 0)&&(ival <= maxprefault)) {
```

```
                parms.pre_fault = ival ;
        }
}
/*
 * postflt -- set postfault recording time
 */
postflt ()
{
        int ival ;

        printf ("Current postfault recording time %d cycles\nEnter new postfault record:
        if((readf("%d",&ival)>0)&&(ival > 0)&&(ival <= MAXPOSTFAULT)) {
                parms.post_fault = ival ;
        }
}
/*
 * wavedsp -- set waveform display format
 */
wavedsp ()
{
        if (ask ("Display on terminal")) {
                parms.daflag = 0 ;
                if (ask ("Graph fault waveforms")) {
                        parms.gflag = 1 ;
                }
                else {
                        parms.gflag = 0 ;
                }
        }
        else {
                parms.daflag = 1 ;
        }
}
/*
 * absinc -- set waveform display abscissa increment
 */
absinc ()
{
        float fval ;

        printf ("Current abscissa increment %.2f degrees ", parms.thetainc) ;
        if (parms.daflag) {
                darate () ;
        }
        printf ("\nEnter new abscissa increment (degrees) (0.5 to 60) : ") ;
        if ((readf ("%f",&fval) > 0) && (fval >= 0.5) && (fval <= 60)) {
                parms.thetainc = fval ;
        }
        if (parms.daflag) {
                darate () ;
        }
        printf ("\n") ;
}

/*
 * darate -- print d/a output rate as seconds per second
```

```
 */
darate ()
{
        printf ("(%.2f seconds per second)",720. / parms.thetainc) ;
}

/*
 * ls -- list current parameters
 */
ls (arg,cmdpntr)
        struct cmd *cmdpntr ;
{
        lsl (&parms, -1, cmdpntr) ;
}

/*
 * lsf -- list parameters for just one fault
 */
lsf (arg,cmdpntr)
        struct cmd *cmdpntr ;
[
        lsl (&entry, fnumber, cmdpntr) ;
}

/*
 * lsl - list parameters
 *    - arguments are:
                        -a - list analog channel parameters
                        -e - list event channel parameters
                        -f - list faults
                        -g - list global parameters
                        -t - print current time
        - if no argument is entered '-gf' is defaulted to.
*/
lsl (directory, afnum, cmdpntr)
 struct direct *directory ;
 struct cmd *cmdpntr ;
{
        extern int argc ;
        extern char *argv[] ;
        extern char *flthed ;
        register char *cpntr ;
        register int i ;
        register struct direct *dentry ;
        int free ;
        int fnum ;

        if (argc > 2) {
                puse (cmdpntr) ;
                return ;
        }
        if (argc == 1) {
                argv[1] = "-gf" ;
        }
        cpntr = argv[1] ;
        if ((*cpntr++ != '-') || (*cpntr == '\0')) {
```

```
                puse (cmdpntr) ;
                return ;
        }
        dentry = directory ;
        for (;;) {
                fnum = afnum ;
                switch (*cpntr++ & 0377) {
                  default :
                        puse (cmdpntr) ;
                        return ;

                  case '\0' :
                        return ;

                  case 'a' :
                        for (i = 0 ; i < dentry->nchan ; i++) {
                                pparms (1, i, dentry) ;
                        }
                        break ;

                  case 'e' :
                        for (i = 0 ; i <= maxechan ; i++) {
                                pparms (0, i, dentry) ;
                        }
                        break ;

                  case 'f' :
                        printf (&flthed) ;
                        if (fnum < 0) {
                                fnum = 0 ;
                                free = NBLOCKS ;
                                for (i = DIRECTORY ; 1 ; i =+ DBLOCKS) {
                                        if ((rx_read (0, i, &entry,
                                            sizeof entry) < 0) ||
                                            (entry.last_block == 0)) {
                                                break ;
                                        }
                                        if (entry.last_block < 0) {
                                                continue ;
                                        }
                                        free =- entry.last_block - entry.first_block ;
                                        lsfault (fnum++) ;
                                }
                                printf ("Room for %d cycles\n",free) ;
                        }
                        else {
                                lsfault (fnum) ;
                        }
                        break ;

                  case 'g' :
                        if (afnum < 0) {
                                printf ("\nRecording Label: \"%s\"\n",dentry->dsname) ;
                                printf ("%3d Prefault Cycles\n%3d Postfault Cycles\n",
                                        dentry->pre_fault, dentry->post_fault) ;
                                printf("%d Hz Sampling Frequency\n%d Analog Channels\n"
```

```
                                        ,dentry->rate,dentry->nchan) ;
                                printf ("%s clock\n",dentry->ext_clock?"External":
                                        "Internal") ;
                        }
                        if (parms.daflag) {
                                printf ("Output on d/a's") ;
                        }
                        else {
                                printf ("Table with%s graph",
                                        parms.gflag ? "" : "out" ) ;
                        }
                        printf (" in %.2f degree steps", parms.thetainc) ;
                        if (parms.daflag) {
                                darate () ;
                        }
                        printf ("\n") ;
                        break ;

                  case 't' :
                        timel (1) ;
                        break ;
                }
                printf ("\n") ;
        }
}

/*
 * rmfault -- remove faults from directory
 */
rmfault (arg, cmdpntr)
        struct cmd *cmdpntr ;
{
        extern int argc ;
        extern char *argv[] ;
        extern noints () ;
        extern reset () ;
        register int i ;
        register int eflag ;
        register int rdblock ;
        int fnum ;
        int faults[20] ;
        int *ipntr ;
        char **cpntr ;
        int nremove ;

        eflag = 0 ;
        do {
                attach (INTERRUPT, &noints, 0) ;
                if (argc < 2) {
                        eflag++ ;
                        break ;
                }
                if (strcmp ("*", argv[1]) == 0) {
                        clear () ;
                        i = 0 ;
                        break ;
```

```
                }
                ipntr = faults ;
                cpntr = &argv[1] ;
                printf ("rm") ;
                nremove = argc - 1 ;
                for (i = nremove ; i ; i--) {
                        if (**cpntr != 'f') {
                                eflag++ ;
                        }
                        **cpntr = '0' ;
                        if (_scanargs ("%d", cpntr, &ipntr) != 1) {
                                eflag++ ;
                        }
                        if (*ipntr < 0) {
                                eflag++ ;
                        }
                        cpntr++ ;
                        printf (" f%d", *ipntr++) ;
                        if (eflag) {
                                goto out ;
                        }
                }
                if (ask ("") == 0) {
                        break ;
                }
                *ipntr = -2 ;
#       ifdef DEBUG
                if (debugf) {
                        for (i = 0 ; ((i < 20) && (faults[i] != -2)) ; i++) {
                                printf ("%d ",faults[i]) ;
                        }
                        printf ("\n") ;
                }
#       endif
                fnum = 0 ;
                i = nremove ;
                for (rdblock = DIRECTORY ; (i) && (rdblock < NBLOCKS) ; rdblock =+ DBLO
#       ifdef DEBUG
                if (debugf) {
                        printf ("fnum %d i %d\n",fnum, i) ;
                }
#       endif
                        if (rx_read (0, rdblock, &entry,
                                (BLK_NUM_SIZE)) < 0) {
                                i = 0 ;
                                break ;
                        }
                        if (entry.last_block == 0) {
                                break ;
                        }
                        if (entry.last_block < 0) {
                                continue ;
                        }
                        for (ipntr = faults ; *ipntr != -2 ; ipntr++) {
                                if (*ipntr == fnum) {
                                        entry.last_block = -1 ;
```

```
                                        if (rx_write (0, rdblock, &entry,
                                                (BLK_NUM_SIZE)) < 0) {
                                                i = 0 ;
                                                break ;
                                        }
                                        *ipntr = -1 ;
                                        i-- ;
                                        break ;
                                }
                        }
                        fnum++ ;
                }
        } while (0) ;
    out :
        if (i) {
                perr ("Bad fault number") ;
        }
        if (eflag) {
                puse (cmdptr) ;
        }
        attach (INTERRUPT, &reset, 0) ;
}

/*
 * puse -- print command usage
 */
puse (acmdptr)
        struct cmd *acmdptr ;
{
        register struct cmd *cmdptr ;

        cmdptr = acmdptr ;
        printf ("Usage: %s %s\n",cmdptr->cmd_name,cmdptr->cmd_options) ;
}

/*
 * clear all directory space
 */
clear ()
{
        register int *ipntr ;
        register int track ;

        if (ask ("Are you sure you want to clear the entire directory")) {
                for (ipntr = adbuf ; ipntr < adbuf+(26 * 256 / 2) ; ) {
                        *ipntr++ = 0 ;
                }
                for (track = DTRACK ; track < NTRACKS ; track++) {
                        rx_write (0, track * 26, adbuf, 26 * 256) ;
                }
        }
}

#ifdef DEBUG
/*
 * turn debugging flag on/off
```

```
 */

debug ()
{
        printf ("debugf changed to %d\n",debugf =^ 1) ;
}

/*
 * dump out a disk block
 */
dump ()
{
        char *drive, *track, *sector ;
        register int i ;
        register int *ipntr ;

        do {
                printf ("drive track sector:") ;
                if (inputf ("%d%d%d",&drive, &track, &sector) < 0) {
                        return ;
                }
        } while ((drive > 1) || (track > 76) || (sector < 1) || (sector > 26));
        if (rx_read (drive, track * 26 + sector - 1, adbuf, 256) < 0) {
                return ;
        }
        i = 8 ;
        for (ipntr = adbuf ; ipntr < adbuf + 128 ; ipntr++) {
                printf ("%8o",*ipntr) ;
                if (--i == 0) {
                        printf ("\n") ;
                        i = 8 ;
                }
        }
}
#endif
```

```
#include <stdio.h>
#include <setjmp.h>
#include "c00.h"

/*
 * recordfault -- record a single fault
 */
recordfault ()
{
        register struct dbuf *routofpntr ;
        int track ;
        register int sector ;
        register char *message ;
        struct tdata prtime ;

        routofpntr = outofpntr ;
        track = ftrack ;
        sector = fsector ;
        message = "\007Recording\r\n" ;
        for (;;) {
                if ((DL11->dl_xcsr < 0) && (*message)) {
                        DL11->dl_xbuf = *message++ ;
                }
                if ((DL11->dl_rcsr < 0) && ((DL11->dl_rbuf & 0177) == CNTLX)) {
                        status = 0 ;
                        return (-1) ;
                }
                if (routofpntr->b_forw == intopntr) {  /* disk has caught up */
#       ifdef DEBUG
                if (debugf) {
                        printf ("caught up\n") ;
                }
#       endif
                        if (status & SAMPLING) {
                                continue ;
                        }
                        else {
                                break ;
                        }
                }
                if ((RX02->rx_csr & RXDONE) == 0) {
                        continue ;
                }
                if (RX02->rx_csr < 0) {
                        rx_panic () ;
                }
                RX02->rx_csr = HI_DENS | FILL_BUFF | RXGO ;
                while (RX02->rx_csrlo >= 0) ;
                RX02->rx_db = 128 ;
                while (RX02->rx_csrlo >= 0) ;
                RX02->rx_db = routofpntr->flt_data ;
                while ((RX02->rx_csr & RXDONE) == 0) ;
                if (RX02->rx_csr < 0) {
                        rx_panic () ;
                }
                RX02->rx_csr = HI_DENS | UNIT_1 | WRITE_SECTOR | RXGO ;
```

```
 */

debug ()
{
        printf ("debugf changed to %d\n",debugf =^ 1) ;
}

/*
 * dump out a disk block
 */
dump ()
{
        char *drive, *track, *sector ;
        register int i ;
        register int *ipntr ;

        do {
                printf ("drive track sector:") ;
                if (inputf ("%d%d%d",&drive, &track, &sector) < 0) {
                        return ;
                }
        } while ((drive > 1) || (track > 76) || (sector < 1) || (sector > 26));
        if (rx_read (drive, track * 26 + sector - 1, adbuf, 256) < 0) {
                return ;
        }
        i = 8 ;
        for (ipntr = adbuf ; ipntr < adbuf + 128 ; ipntr++) {
                printf ("%8o",*ipntr) ;
                if (--i == 0) {
                        printf ("\n") ;
                        i = 8 ;
                }
        }
}
#endif
```

```
#include <stdio.h>
#include <setjmp.h>
#include "c00.h"

/*
 * recordfault -- record a single fault
 */
recordfault ()
{
        register struct dbuf *routofpntr ;
        int track ;
        register int sector ;
        register char *message ;
        struct tdata prtime ;

        routofpntr = outofpntr ;
        track = ftrack ;
        sector = fsector ;
        message = "\007Recording\r\n" ;
        for (;;) {
                if ((DL11->dl_xcsr < 0) && (*message)) {
                        DL11->dl_xbuf = *message++ ;
                }
                if ((DL11->dl_rcsr < 0) && ((DL11->dl_rbuf & 0177) == CNTLX)) {
                        status = 0 ;
                        return (-1) ;
                }
                if (routofpntr->b_forw == intopntr) {  /* disk has caught up */
#       ifdef DEBUG
                if (debugf) {
                        printf ("caught up\n") ;
                }
#       endif
                        if (status & SAMPLING) {
                                continue ;
                        }
                        else {
                                break ;
                        }
                }
                if ((RX02->rx_csr & RXDONE) == 0) {
                        continue ;
                }
                if (RX02->rx_csr < 0) {
                        rx_panic () ;
                }
                RX02->rx_csr = HI_DENS | FILL_BUFF | RXGO ;
                while (RX02->rx_csrlo >= 0) ;
                RX02->rx_db = 128 ;
                while (RX02->rx_csrlo >= 0) ;
                RX02->rx_db = routofpntr->flt_data ;
                while ((RX02->rx_csr & RXDONE) == 0) ;
                if (RX02->rx_csr < 0) {
                        rx_panic () ;
                }
                RX02->rx_csr = HI_DENS | UNIT_1 | WRITE_SECTOR | RXGO ;
```

```
                while (RX02->rx_csrlo >= 0) ;
                RX02->rx_db = sector ;
                while (RX02->rx_csrlo >= 0) ;
                RX02->rx_db = track ;

                if (++sector > 26) {
                        sector = 1 ;
                        if (++track > 76) {
                                status =& ~SAMPLING ;
                                break ;
                        }
                }
                routofpntr = routofpntr->b_forw ;
                outofpntr = routofpntr ; /* update outofpntr so overrun */
                                         /* checking works */
        }
        parms.first_block = fblock ;
        parms.last_block = track * 26 + sector - 1 ;
        parms.post_fault = nrecord - nleft ;
#       ifdef DEBUG
        if (debugf) {
        printf ("block %d to %d\n",parms.first_block,parms.last_block) ;
        }
#       endif
        printf ("%s%d Cycles Recorded -- ", message,
                 parms.last_block-parms.first_block) ;
        unpack (&parms.fault_time, &prtime) ;
        printf ("%3d  %2d:%02d:%02d:%04d\n", prtime.c_day, prtime.c_hr,
          prtime.c_min, prtime.c_sec, sector = (prtime.c_ticks * 10.416667)) ;
        printf ("Room for %d more cycles\n",NBLOCKS-parms.last_block) ;
        return (rx_write (0, dblock, &parms, sizeof parms)) ;
}

/*
 * findfreeblock -- set up pointers onto the disk
 */
findfreeblock ()
{
        register int iblock ;
        register int oblock ;
        register int nblocks ;
        int ntrans ;
        int sentmessage ;

        dblock = DIRECTORY ;
        fblock = 0 ;
        sentmessage = 0 ;
        for (;;) {
                if (rx_read (0, dblock, &entry, SECTORSIZE) < 0) {
                        return (-1) ;
                }
                if (entry.last_block == 0) {
                        break ;
                }
                if (entry.last_block < 0) {
                        if (squishdir() < 0) {
```

```
                        return (-1) ;
                    }
                    continue ;
                }
                if (fblock != entry.first_block) {
                    if (sentmessage == 0) {
                        printf ("Squishing data\n") ;
                        sentmessage++ ;
                    }
                    nblocks = entry.last_block - entry.first_block ;
                    iblock = entry.first_block ;
                    oblock = fblock ;
                    while (nblocks) {
                        ntrans = nblocks>maxcycles?maxcycles:nblocks ;
                        nblocks -= ntrans ;
                        if ((rx_read (1, iblock, adbuf, ntrans*SECTORSIZE) < 0)
                        || (rx_write (1, oblock, adbuf, ntrans*SECTORSIZE) < 0)
                            return (-1) ;
                        }
                        oblock += ntrans ;
                        iblock += ntrans ;
                    }
                    entry.first_block = fblock ;
                    entry.last_block = oblock ;
                    rx_write (0, dblock, &entry, SECTORSIZE) ;
                }
                fblock = entry.last_block ;
                if ((dblock += DBLOCKS) >= NBLOCKS) {
                    break ;
                }
            }
            if (((fblock + parms.pre_fault + 1) >= NBLOCKS)||(dblock >= NBLOCKS)) {
                printf ("\n******** \007DISK FULL\007 *******\n") ;
                return (-1) ;
            }
            if (rx_write (1, fblock, &entry, 2) < 0) {      /* position head */
                return (-1) ;
            }
#           ifdef DEBUG
            if (debugf) {
            printf ("dblock:%d fblock:%d\n",dblock,fblock) ;
            }
#           endif

            ftrack = fblock / 26 ;
            fsector = (fblock % 26) + 1 ;
            return (0) ;
}

/*
 * lsfault -- list particulars of desired fault
 */
lsfault (faultnum)
{
            struct tdata prtime ;
            register int i ;
```

```
        if (entry.last_block <= 0) {
                return ;
        }
#       ifdef DEBUG
        if (debugf) {
                printf ("%d %d %d",dblock, entry.first_block, entry.last_block) ;
        }
#       endif
        unpack (&entry.fault_time, &prtime) ;
        printf ("%2d  %4d  %3d  %3d  %3d %3d %2d:%02d:%02d.%04d %c  \"%.32s\"\n",
                faultnum, entry.last_block-entry.first_block,entry.pre_fault,
                entry.post_fault,
                entry.rate,
                prtime.c_day, prtime.c_hr, prtime.c_min, prtime.c_sec,
                i = (prtime.c_ticks * 10.416667),
                entry.ext_clock ? 'E' : 'I' ,
                entry.dsname ) ;
}

/*
 * rmlfault -- remove fault
 */
rmlfault ()
{
        extern noints() ;
        extern reset () ;
        char *i ;
        register int npost ;
        register int npreoff ;
        register int npostoff ;

        npreoff = 0 ;
        npostoff = 0 ;
        if (ask ("Remove Entire Fault")) {
                npostoff = entry.last_block+1 ;
        }
        else {
                printf ("Number of prefault cycles to remove (0 to %d) ?",
                        entry.pre_fault-1) ;
                if (inputf ("%d",&i) < 0) {
                        return ;
                }
                if (i < entry.pre_fault) {
                        npreoff = i ;
                }
                else {
                        perr ("Number out of range") ;
                        return ;
                }
                npost = entry.last_block-entry.first_block-entry.pre_fault ;
                printf ("Number of postfault cycles to remove (0 to %d) ?",
                        npost-1) ;
                if (inputf ("%d",&i) < 0) {
                        return ;
                }
```

```
                if (i < npost) {
                        npostoff = i ;
                }
                else {
                        perr ("Number out of range") ;
                        return ;
                }
        }
        attach (INTERRUPT, &noints, 0200) ;
        entry.first_block =+ npreoff ;
        entry.pre_fault =- npreoff ;
        entry.last_block =- npostoff ;
        entry.post_fault =- npostoff ;
        if (entry.post_fault < 0) {
                entry.post_fault = 0 ;
        }
        if (rx_write (0, dblock, &entry, sizeof entry) < 0) {
                rx_read (0, dblock, &entry, sizeof entry) ;
        }
        attach (INTERRUPT, &reset, 0) ;
}

/*
 * edfault -- interactively display and delete faults
 */
edfault (arg, cmdpntr)
        struct cmd *cmdpntr ;
{
        extern int argc ;
        extern char *argv[] ;
        extern char *flthed ;
        extern struct cmd *edcmds ;
        register char *i ;
        char prompt[10] ;

        if ((argc != 2) || (argv[1][0] != 'f') || (argv[1][1] == 0)) {
                puse (cmdpntr) ;
                return ;
        }
        argv[1][0] = '0' ;
        if (_IGetint (argv[1], &fnumber, 10) < 0) {
                printf ("Bad number") ;
        }
        dblock = DIRECTORY - DBLOCKS ;
        for (i = 0 ; i <= fnumber ; ) {
                dblock =+ DBLOCKS ;
                if ((rx_read (0, dblock, &entry, sizeof
                        entry) < 0) || (entry.last_block == 0)) {
                        perr ("Bad Fault Number") ;
                        return ;
                }
                if (entry.last_block > 0) {
                        i++ ;
                }
        }
        printf (&flthed) ;
```

```
        lsfault (fnumber) ;
        sprintf (prompt, "\ned f%d>", fnumber) ;
        while ((entry.last_block >= 0) && (shell (prompt, &edcmds, 0))) ;
}

/*
 * squishdir -- reorganize directory
 */
squishdir ()
{
        register int idblock ;
        register int odblock ;
        register int *ipntr ;
#       ifdef DEBUG
        int sdbflg ;
#       endif

        printf ("Squishing directory\n") ;
#       ifdef DEBUG
        if (debugf) {
        printf ("dblock %d  fblock %d  entry.last_block %d\n",dblock,fblock,
                entry.last_block) ;
        }
#       endif
        for (ipntr = adbuf ; ipntr < adbuf + 128 ; ) {
                *ipntr++ = 0 ;
        }
        idblock = odblock = dblock ;
        do {
                while (entry.last_block < 0) {
                        idblock =+ DBLOCKS ;
                        if (rx_read (0, idblock, &entry, sizeof entry) < 0) {
                                return (-1) ;
                        }
                }
                while (entry.last_block > 0) {
                        if (rx_write (0, odblock, &entry, sizeof entry) < 0) {
                                return (-1) ;
                        }
                        idblock =+ DBLOCKS ;
                        odblock =+ DBLOCKS ;
                        if (rx_read (0, idblock, &entry, sizeof entry) < 0) {
                                return (-1) ;
                        }
                }
        } while (entry.last_block) ;
#       ifdef DEBUG
        if (debugf) {
                printf ("odblock %d\n",odblock) ;
        }
        sdbflg = debugf ;
        debugf = 0 ;
#       endif
        while (odblock < NBLOCKS) {
                if (rx_write (0, odblock++, adbuf, SECTORSIZE) < 0) {
                        return (-1) ;
```

```
                }
        }
#       ifdef DEBUG
        debugf = sdbflg ;
#       endif
        return (0) ;
}

/*
 * prfault -- print out fault waveforms
 */
prfault (arg, cmdpntr)
        struct cmd *cmdpntr ;
{
        extern int argc ;
        extern char *argv[] ;
        register int *ipntr ;
        register int *opntr ;
        register int i ;
        float interpolate () ;
        int sampnum ;
        float fval ;
        float incval ;
        int ival ;
        int firsttime ;
        int bit ;
        int block ;
        int nblocks ;
        int ntrans ;
        int count ;
        int chan ;
        char *begin ;
        char c ;
        struct tdata ftime ;
        struct tdata prtime ;

        if ((argc != 2) || (((c = argv[1][0]) != 'a') && (c != 'e'))) {
                puse (cmdpntr) ;
                return ;
        }
        if (rx_read (1, entry.first_block + entry.pre_fault, adbuf, SECTORSIZE) < 0) {
                return ;
        }

        if ( *(adbuf + 133 - entry.nchan) < 0) {        /* flag word of last sample in
                printf ("Bad Data -- Freeze flag not found\n") ;
                return ;
        }
        sampnum = 1 - (entry.smpcyc * (entry.pre_fault + 1)) ;
        firsttime = 1 ;
        argv[1][0] = '0' ;
        chan = -1 ;
        chan = atoi (argv[1]) ;
        if (c == 'e') {
                printf ("\n     Time       e15 e14 e13 e12 e11 e10  e9   e8   e7   e6   e5
                nblocks = entry.last_block - entry.first_block ;
```

```
                block = entry.first_block ;
                unpack (&entry.fault_time, &ftime) ;
                while (nblocks) {
                        ntrans = nblocks > maxcycles ? maxcycles : nblocks ;
                        nblocks -= ntrans ;
                        if (rx_read (1, block, adbuf, ntrans * SECTORSIZE) < 0) {
                                return ;
                        }
                        block += ntrans ;
                        for (ipntr = adbuf ; ipntr < adbuf + (ntrans * 128) ; ipntr +=
                                if (firsttime) {
                                        sampno = (ipntr[4] & ~DTMASK) + 010000 ;
                                }
                                if ((ipntr[4] & ~DTMASK) != (sampno += 010000)) {
                                        printf ("Non-consecutive sample\n") ;
                                        return ;
                                }
                                i = (ipntr[0] & 0170000)
                                        | ((ipntr[3] >> 4) & 07400)
                                        | ((ipntr[1] >> 8) & 0360)
                                        | ((ipntr[2] >> 12) & 017) ;
                                if ((i != lastdig) || (firsttime)) {
                                        firsttime = 0 ;
                                        fixtime (&ftime, &prtime,
                                           sampnum*((entry.rate==960)?1:2)) ;
                                        printf ("%3d %2d:%02d:%02d:%04d ",
                                                prtime.c_day,
                                                prtime.c_hr,
                                                prtime.c_min,
                                                prtime.c_sec,
                                                ival=prtime.c_ticks*10.41667) ;
                                        for (bit = 0100000 ; bit ;
                                                bit = (bit >> 1) & 077777) {
                                                printf ("%c",
                                                        bit & i ? '0' : '1') ;
                                                if (bit != 1) {
                                                        printf ("   ") ;
                                                }
                                        }
                                        printf ("\n") ;
                                }
                                ++sampnum ;
                                lastdig = i ;
                        }
                }
                return ;
        }
        if ((chan < 0) || (chan >= entry.nchan)) {
                perr ("Bad Channel") ;
                return ;
        }
        block = entry.last_block - entry.first_block ;
        pparms (1, chan, &entry) ;
        do {
                printf ("\nFirst Cycle   Number of cycles to display : ") ;
                if (inputf ("%d%f",&begin,&fval) < 0) {
```

```
                        return ;
                }
                count = roundup (fval) ;
                if ((fval > maxdisplay) || (fval < 0.1)) {
                        count = -1 ;
                }
        } while ((begin >= block) || (begin + count > block) || (count <= 0)) ;
        block = entry.first_block + begin ;
        if (begin != 0) {
                block-- ;
        }
        count =+ 2 ;
        opntr = adbuf ;
        do {
                if (block == entry.last_block) {
                        block-- ;
                }
                ipntr = adend - 128 ;
                if (rx_read (1, block++, ipntr, SECTORSIZE) < 0) {
                        return ;
                }
                if ((firsttime) && (begin == 0)) {
                        block-- ;
                        firsttime = 0 ;
                }
                i = entry.smpcyc ;
                do {
                        *opntr++ = (*(ipntr + chan) & DTMASK) - DTOFFSET ;
#       ifdef DEBUG
        if (debugf) {
                        printf ("%d\n",*--opntr) ;
                                opntr++ ;
        }
#       endif
                        ipntr =+ entry.nchan ;
                } while (--i) ;
        } while (--count) ;
        ipoint = entry.smpcyc ;
        incval = 360. / entry.smpcyc ;
        lastpoint = incval - ((entry.pre_fault - (i = begin) + 1) * 360.) ;
        nextpoint = lastpoint + incval ;
        fval = fval * 360 + lastpoint ;
        graphchan (lastpoint, fval, entry.smpcyc) ;
}

/*
 * fixtime - add signed offset to current time
 */
fixtime (from, to, offset)
        struct tdata *from ;
        struct tdata *to ;
{
        register struct tdata *fpntr ;
        register struct tdata *tpntr ;
        register int i ;
```

```
#       ifdef DEBUG
        if (debugf) {
                printf ("fixtime -- offset %d ticks\n", offset) ;
        }
#       endif
        fpntr = from ;
        tpntr = to ;
        i = offset ;

        tpntr->c_ticks = fpntr->c_ticks ;
        tpntr->c_day = fpntr->c_day ;
        tpntr->c_hr = fpntr->c_hr ;
        tpntr->c_min = fpntr->c_min ;
        tpntr->c_sec = fpntr->c_sec ;

        tpntr->c_ticks =+ i % 960 ;
        i =/ 960 ;
        tpntr->c_sec =+ i ;
        if (tpntr->c_ticks < 0) {
                tpntr->c_ticks =+ 960 ;
                tpntr->c_sec-- ;
        }
        if (tpntr->c_ticks >= 960) {
                tpntr->c_ticks =- 960 ;
                tpntr->c_sec++ ;
        }
        if (tpntr->c_sec < 0) {
                tpntr->c_sec =+ 60 ;
                tpntr->c_min-- ;
        }
        if (tpntr->c_sec >= 60) {
                tpntr->c_sec =- 60 ;
                tpntr->c_min++ ;
        }
        if (tpntr->c_min < 0) {
                tpntr->c_min =+ 60 ;
                tpntr->c_hr-- ;
        }
        if (tpntr->c_min >= 60) {
                tpntr->c_min =- 60 ;
                tpntr->c_hr++ ;
        }
        if (tpntr->c_hr < 0) {
                tpntr->c_hr =+ 24 ;
                tpntr->c_day-- ;
        }
        if (tpntr->c_hr >= 24) {
                tpntr->c_hr =- 24 ;
                tpntr->c_day++ ;
        }
}
```

```
/*
/* Assembler support routines for the Fault Data Recorder
/*

/* debugging flag
debug = 0


/* non-unix instructions
rti = 2 ^ clc
mtps = 106400 ^ clr

/* C routines accessed by this segment
.globl _monitor,_rtcint,_panic

/* C variables accessed by this segment
.globl _adpntr,_nbsamp,_adend,_adbuf,_lastpntr,_dtrig,_hnbsamp
.globl _leadsig,_trailsig,_lastdig,_sampno,_nrecord,_nleft,_reenter
.globl _status,_passno,_allchan,_adcounter,_smpblk,_intopntr,_outofpntr
.globl _fltdet

/ adpntr        -- pointer into data buffer
/ nbsamp        -- number of bytes per sample
/ adend         -- end of data buffer
/ adbuf         -- data buffer
/ lastpntr      -- pointer to data samples one cycle back
/ dtrig         -- delta trigger limits
/ hnbsamp       -- vector of offsets into adbuf
/ leadsig       -- fault condition when these bits go hi
/ trailsig      -- fault condition when these bits go lo
/ lastdig       -- previous digital sample
/ sampno        -- sample number
/ nrecord       -- number of disk blocks to record after fault clears
/ nleft         -- number of disk blocks left to record
/ reenter       -- flag indicating overrun error
/ status        -- system status word
/ passno        -- pass number
/ allchan       -- all channels flag
/ adcounter     -- another pass counter
/ smpblk        -- number of samples per disk block
/ intopntr      -- pointer into adbuf
/ outofpntr     -- pointer out of adbuf
/ fltdet        -- fault detection flag


/ analog to digital converter parameters
/ BOARD MUST BE SET UP FOR OFFSET BINARY
/ HI ORDER BITS MUST ALWAYS BE 1's
DTCSRLO = 177000
DTCSRHI = 177001
DTDBUF  = 177002
DTWCNT  = 177004
DTBADD  = 177006
GO = 1
DMA_EN = 02
ADCHECK = 040200
```

```
/ satellite clock parameters
CLOCKCSR = 0174410

/ dr11 parallel line parameters
DRXBUF = 0174402
DRRBUF = 0174404

/ external clock parameters
CLKCSR = 0174410
CLKRLO = 0174414
CLKRHI = 0174424

/ dlv11j serial line clock parameters
DLXBUF = 0176526

/ offsets into directory structure
EXT_CLK = 4
TBUF = 012

/ other constants
SAMPLING = 01   / status word bit
RECORDING = 02  / status word bit


/*****************************************************************************
/* _monitor -- read data, check for fault condition
/*
/*
_monitor:
/* Initialization

        .if debug
          mov     $3777,*$DTDBUF          / timer
        .endif
          tst     *$_reenter              / overrun??
          beq     1f                      / no, just continue
          mov     $errmsg,-(sp)           / yes, print message
          jsr     pc,*$_panic
1:
          inc     *$_reenter              / put up flag
          bit     $SAMPLING,*$_status     / are we really taking data?
          bne     1f                      / yes, go read data
          clr     *$_reenter              / otherwise pull down flag
          rti                             / and return
1:
          mov     r5,-(sp)                / save the registers
          mov     r4,-(sp)
          mov     r3,-(sp)
          mov     r2,-(sp)
          mov     r1,-(sp)
          mov     r0,-(sp)
          bit     $RECORDING,*$_status    / recording?
          bne     1f                      / yes, don't change time
          mov     $_ctime,r0              / point r0 to current time
          mov     $_parms+TBUF,r1         / point r1 to fault time
```

```
        mov     (r0)+,(r1)+             / copy time
        mov     (r0)+,(r1)+
        mov     (r0)+,(r1)+
        mov     (r0),(r1)
1:
        cmp     $ADCHECK,*$DTCSRLO      / see if dma went correctly
        beq     1f                      / if the same, we are o.k.
        mov     $fltmsg,-(sp)           / otherwise it is a calamity
        jsr     pc,*$_panic
                                        / this should actually not be this
                                        / serious, but it is hard to recover
                                        / from.
                                        / Actually given the time constraints
                                        / it is impossible to recover from
1:
        mov     *$_nbsamp,r3            / get number of bytes per sample
        mov     *$_adpntr,r0            / get current a/d pointer
        mov     r0,r1                   / save it in r1
        mov     r0,-(sp)                / and on the stack
        mov     *$DRRBUF,-(sp)          / get digital data
        add     r3,r0                   / add in number of bytes per sample
        cmp     *$_adend,r0             / end of buffer?
        jhi     1f                      / no, skip ahead
        mov     *$_adbuf,r0             / yes, wrap around
1:
        mov     r0,*$DTBADD             / set up a/d bus address
        mov     r0,*$_adpntr            / save current pointer
        mov     $!16.,*$DTWCNT          / set up word count, always 16 channels
                                        / since the hardware is hard to set to
                                        / channel 0 otherwise.

        clr     r5                      / clear fault detection flag
        mtps    r5                      / and turn interrupts back on

        mov     *$_lastpntr,r0          / get pointer to last cycle
        add     r3,r0                   / addin number of bytes per sample
        cmp     *$_adend,r0             / end of buffer?
        jhi     1f                      / no, skip ahead
        mov     *$_adbuf,r0             / yes, wrap around
1:
        mov     r0,*$_lastpntr          / save updated pointer
        mov     $170000,r4              / set up bit mask

        sub     $2,*$_passno            / update pass number
        bpl     L1                      / if >= 0, do analog trigger
/*
/* Digital triggering routine                          Patched.
/*                                                     E. Norum
        mov     (sp),r0                 / get digital data  Aug. 7/80
        mov     *$_lastdig,r3           / get data from last sample
        xor     r0,r3                   / any bits changed?
        beq     8f                      / no, skip ahead
        mov     r0,r2                   / yes, save a copy of data
        com     r2                      / get complement of digital data
        com     r3                      / set mask of bits that changed
        bic     r3,r0                   / mask for bits that went hi
```

```
        bic     r3,r2                   / mask for bits that went low
        bit     *$_leadsig,r0           / did bit go hi?
        bne     9f                      / yes, go put up flag
        bit     *$_trailsig,r2          / did bit go lo
        beq     8f                      / no, skip ahead
9:
        inc     r5                      / put up flag
8:
        mov     r0,*$_lastdig           / save current digital state
        mov     $10,*$_passno           / reset pass number
        jbr     done                    / and go to done

/*
/* Analog triggering routine
/*
L1:
        mov     *$_passno,r2            / get pass number
        mov     _hnbsamp(r2),r3         / get offset
        mov     $_dtrig,r2              / get pointer to trigger values
        add     r3,r0                   / set pointers to offset
        add     r3,r1
        add     r3,r2
        tst     *$_allchan              / all channels?
        beq     do2b                    / no, do only half

        mov     (r0)+,r3                / get last sample
        bis     r4,r3                   / turn on top bits
        sub     (r1)+,r3                / subtract this cycles value
        bpl     1f                      / if positive, skip
        neg     r3                      / get absolute value
1:
        cmp     (r2)+,r3                / check difference against setpoint
        adc     r5                      / set flag if difference is too large

        mov     (r0)+,r3                / get last sample
        bis     r4,r3                   / turn on top bits
        sub     (r1)+,r3                / subtract this cycles value
        bpl     1f                      / if positive, skip
        neg     r3                      / get absolute value
1:
        cmp     (r2)+,r3                / check difference against setpoint
        adc     r5                      / set flag if difference is too large

do2b:
        mov     (r0)+,r3                / get last sample
        bis     r4,r3                   / turn on top bits
        sub     (r1)+,r3                / subtract this cycles value
        bpl     1f                      / if positive, skip
        neg     r3                      / get absolute value
1:
        cmp     (r2)+,r3                / check difference against setpoint
        adc     r5                      / set flag if difference is too large

        mov     (r0)+,r3                / get last sample
        bis     r4,r3                   / turn on top bits
        sub     (r1)+,r3                / subtract this cycles value
```

move line

```
        bpl     1f                      / if positive, skip
        neg     r3                      / get absolute value
1:
        cmp     (r2)+,r3                / check difference against setpoint
        adc     r5                      / set flag if difference is too large

done:

/*
/* Pack digital data in with analog data
/*

        mov     (sp)+,r0                / get digital data
        com     r4                      / switch bit mask around
        mov     (sp)+,r1                / point r1 to current data samples
        mov     r0,r2                   / copy digital data
        bic     r4,r2                   / mask for hi bits
        bic     r2,(r1)+                / stuff in 4 bits
        swab    r0                      / switch around digital data
        mov     r0,r2                   / copy it again
        bic     r4,r2                   / mask for 4 bits
        bic     r2,(r1)+                / stuff in 4 bits
        asl     r0                      / shift digital data up
        asl     r0
        asl     r0
        asl     r0
        mov     r0,r2                   / copy it again
        bic     r4,r2                   / mask for 4 bits
        bic     r2,(r1)+
        swab    r0                      / get last 4 bits
        bic     r4,r0                   / mask for them
        bic     r0,(r1)+                / and stuff them in

/*
/* Update sample number
/*

        add     $010000,*$_sampno       / increment sample number
        bic     *$_sampno,(r1)+         / stuff in sample number

/*
/* Take appropriate action on fault condition
/*

        add     r5,*$_fltdet            / remember if fault was detected
        dec     *$_adcounter            / decrement pass count
        bne     9f                      / if not zero, just return

        mov     *$_smpblk,*$_adcounter  / reset pass counter
        mov     *_intopntr,*$_intopntr  / update into pointer
        cmp     *$_intopntr,*$_outofpntr/ check for overrun
        bne     1f
        mov     $dermsg,-(sp)           / report overrun
        jsr     pc,*$_panic
1:
        bit     $RECORDING,*$_status    / are we recording?
```

```
        beq     8f                      / no, skip ahead

        tst     *$_fltdet               / fault in this block?
        beq     6f                      / no, skip ahead

        mov     *$_nrecord,*$_nleft     / reset number of blocks left
        br      7f                      / and skip ahead
6:
        dec     *$_nleft                / decrement blocks left to record
        bgt     9f                      / if not done, return
        bic     $SAMPLING,*$_status     / otherwise turn off sampling
        br      9f                      / and return
8:
        tst     *$_fltdet               / fault in this block?
        bne     1f                      / yes, go freeze time, etc

        mov     *_outofpntr,*$_outofpntr/ otherwise update out of pointer
        br      9f                      / and return
1:
        bis     $RECORDING,*$_status    / start recording
        bic     $100000,(r1)            / mark sample
        mov     *$_nrecord,*$_nleft     / set up minimum number of blocks
7:
        clr     *$_fltdet               / clear flag
9:

/*
/* Restore registers and return
/*
        mov     (sp)+,r0
        mov     (sp)+,r1
        mov     (sp)+,r2
        mov     (sp)+,r3
        mov     (sp)+,r4
        mov     (sp)+,r5
      .if debug
        mov     $0,*$DTDBUF
      .endif
        clr     *$_reenter              / pull down flag
        rti                             / and return

errmsg: <overrun error\0>
fltmsg: <a/d converter failure\0>
dermsg: <disk overrun\0>
.even

/*****************************************************************************
/* _rtcint -- real time clock interrupt service routine
/*
/* If status word indictates that sampling is in progress, start a/d
/* converter.
/* Put status word on digital output

_rtcint:
        clr     *$DLXBUF                / reprime clock
        add     $1,*$_ctime+6           / increment offset
```

```
        adc     *$_ctime+4
        dec     *$_ticks                / decrement tick count
        bne     8f                      / go return if not ready
        mov     *$_status,*$DRXBUF      / send status word
        mov     *$_iticks,*$_ticks      / reset tick count
        dec     *$_rtticks              / for real-time routines
        bit     $SAMPLING,*$_status     / sampling?
        beq     1f                      / no, just skip ahead
        bisb    $DMA_EN,*$DTCSRLO       / yes, start up a/d
1:
        tst     *$_parms+EXT_CLK        / external clock?
        beq     9f                      / no, go return
        tstb    *$CLKCSR                / external time ready?
        bpl     9f                      / no, go return
        mov     r0,-(sp)                / save r0
        mov     $_ctime,r0              / point r0 to current time
        mov     *$CLKRHI,(r0)+          / get base time
        mov     *$CLKRLO,(r0)+
        clr     (r0)+                   / clear offset
        clr     (r0)
        mov     (sp)+,r0                / restore r0
9:
8:
        rti                             / and return

/
/ C routine unpack
/
/ unpack (from, to)
/ struct tbuf *from ;
/ struct tdata *to ;
/ convert raw time in from to cooked time in to
.globl _unpack
_unpack:
        jsr     r5,*$csv                / save registers
        mov     4(r5),r2                / point r2 to 'from'
        mov     6(r5),r4                / point r4 to 'to'
        mov     (r2)+,r0                / get base time
        mov     (r2)+,r1                / lo order word too
        mov     r1,r3                   / get lo word
        bic     $!17,r3                 / mask for seconds
        mov     r3,(sp)                 / save unit seconds
        ashc    $-4,r0                  / get 10's seconds to lo bits
        mov     r1,r3                   / get lo word
        bic     $!7,r3                  / mask for 10's seconds
        mul     $10.,r3                 / 10's
        add     (sp),r3
        movb    r3,(r4)+                / save seconds
        ashc    $-3,r0                  / get minutes to lo order bits
        mov     r1,r3                   / get lo word
        bic     $!17,r3                 / mask for minutes
        mov     r3,(sp)                 / save unit minutes
        ashc    $-4,r0                  / get 10's minutes to lo bits
        mov     r1,r3                   / get lo word
        bic     $!7,r3                  / mask for 10's minutes
        mul     $10.,r3                 / 10's
```

```
        add     (sp),r3
        movb    r3,(r4)+        / save minutes
        ashc    $-3,r0          / get hours to lo order bits
        mov     r1,r3           / get lo word
        bic     $!17,r3         / mask for hours
        mov     r3,(sp)         / save unit hours
        ashc    $-4,r0          / get 10's hours to lo bits
        mov     r1,r3           / get lo word
        bic     $!3,r3          / mask for 10's hours
        mul     $10.,r3         / 10's
        add     (sp),r3
        mov     r3,(r4)+        / save hours
        ashc    $-2,r0          / get days
        mov     r1,r3           / get lo word
        bic     $!17,r3
        mov     r3,(sp)         / save unit days
        ashc    $-4,r0          / get 10's days in lo bits
        mov     r1,r3           / get 10's days
        bic     $!17,r3         / mask for bits
        mul     $10.,r3
        add     r3,(sp)
        ashc    $-4,r0
        bic     $!3,r1
        mul     $100.,r1
        add     (sp),r1
        mov     r1,(r4)+
/ convert offset and add it in
        mov     (r2)+,r0        / get offset
        mov     (r2),r1
        mov     $960.,(sp)      / ticks per second
        mov     r1,-(sp)        / offset
        mov     r0,-(sp)
        jsr     pc,*$__lurem    / get remainder
        mov     r0,(r4)+        / save ticks
        jsr     pc,*$__ludiv    / get quotient
        mov     (sp)+,(sp)+     / clean up stack
        mov     $60.,(sp)       / seconds per minute
        mov     r1,-(sp)        / offset
        mov     r0,-(sp)
        jsr     pc,*$__lurem    / get remainder
        movb    r0,(r4)+        / save seconds
        jsr     pc,*$__ludiv    / get quotient
        mov     (sp)+,(sp)+     / clean up stack
        mov     $60.,(sp)       / minutes per hour
        mov     r1,-(sp)        / offset
        mov     r0,-(sp)
        jsr     pc,*$__lurem    / get remainder
        movb    r0,(r4)+        / save minutes
        jsr     pc,*$__ludiv    / get quotient
        mov     (sp)+,(sp)+     / clean up stack
        mov     $24.,(sp)       / hours per day
        mov     r1,-(sp)        / offset
        mov     r0,-(sp)
        jsr     pc,*$__lurem    / get remainder
        mov     r0,(r4)+        / save hours
        jsr     pc,*$__ludiv    / get quotient
```

Replace line

```
        mov     (sp)+,(sp)+             / clean up stack
        mov     r0,(r4)                 / save days ← mov r1,(r4)
/ add base time to offset
        mov     6(r5),r0                / point r0 to base time
        mov     r0,r1
        add     $10,r1                  / point r1 to offset seconds
        movb    (r0)+,r2                / get base seconds
        movb    (r1),r3                 / get offset seconds
        add     r3,r2                   / add them together
        movb    r2,(r1)+                / save them
        movb    (r0)+,r2                / get base minutes
        movb    (r1),r3                 / get offset minutes
        add     r3,r2                   / add them together
        movb    r2,(r1)+                / save them
        movb    (r0)+,r2                / get base hours
        movb    (r1),r3                 / get offset hours
        add     r3,r2                   / add them together
        movb    r2,(r1)+                / save them
        inc     r0                      / skip over hole in structure
        inc     r1
        add     (r0)+,(r1)+             / add days together
/ 'carry'
        tst     (r0)+                   / skip past ticks
        cmpb    $60.,(r0)+              / carry over to minute?
        bhi     1f                      / no, skip
        movb    -(r0),r2                / get seconds
        sub     $60.,r2                 / take out one minute
        movb    r2,(r0)+                / put seconds back
        incb    (r0)                    / and bump minutes
1:
        cmpb    $60.,(r0)+              / carry over to hour?
        bhi     1f                      / no, skip
        movb    -(r0),r2                / get minutes
        sub     $60.,r2                 / take out one hour
        movb    r2,(r0)+                / put minutes back
        incb    (r0)                    / and bump hours
1:
        cmpb    $24.,(r0)+              / carry over to day
        bhi     1f                      / no, skip
        movb    -(r0),r2                / get hours
        sub     $24.,r2                 / take out one day
        movb    r2,(r0)+                / put hours back
        inc     r0                      / skip over hole
        inc     (r0)                    / and bump days
1:
        jmp     *$cret                  / and return
```

Patched
E. Norum
Aug 7 /80

```
////////////////////////////////////////////////////////////////////////
//                                                                    //
//                      RX02 BOOTSTRAP                                //
//                                                                    //
////////////////////////////////////////////////////////////////////////

////////////////////////////////////////////////////////////////////////
// Note: This bootstrap must be stored on track 1 sector 1 and track 1
//       sector 3
////////////////////////////////////////////////////////////////////////

////////////////////////////////////////////////////////////////////////
//  CONSTANTS
RXCSR   = 0177170       / RX02 Command Status Register
RXDBUF  = 0177172       / RX02 Data Register
RX02_MD = 04000         / RX02 indication
HI_DENS = 0400          / Hi density (256 bytes per sector)
RX_DONE = 040           / Controller ready for command
EMPTY   = 02            / Empty buffer
READ    = 06            / Read sector
GO      = 01            / Start Command
INIT_DONE = 04          / Initialize done flag

STRACK  = 1             / Track where system starts
SSECTOR = 5             / Sector where system starts

MAGIC   = 0407          / Header magic number
FSECTOR = 1             / First sector on track
LSECTOR = 26.           / Last sector on track
SECSIZW = 128.          / 128 words per sector

BR7     = 0340          / Processor priority level 7
////////////////////////////////////////////////////////////////////////

////////////////////////////////////////////////////////////////////////
// ERROR TERMINATION
// If some error occurs during booting, the processor will halt.
// An error code will be left in register 0.
// The error codes and their meanings are:
// 0. Disk system is not configured as RX02
// 1. First word of system is not MAGIC.
// 2. Timeout occurred while waiting to transfer parameter to disk controller
// 3. Disk error occurred
// 4. System on disk is too big to fit in memory
////////////////////////////////////////////////////////////////////////

////////////////////////////////////////////////////////////////////////
// NON UNIX INSTRUCTIONS
halt = 0 ^ clc
nop = 0240 ^ clc
////////////////////////////////////////////////////////////////////////

////////////////////////////////////////////////////////////////////////
// Start off routine
// S1. Size and clear memory
// 2. Copy program to high memory
```

```
// 3. Start program in high memory

.=0^.                           / start of memory
        nop                     / no operation (DSD-440 Bootstrap needs to
                                / see this as first instruction of program
        br      1f              / branch over trap vector

        trapd ; BR7             / Bus timeout trap vector

1:
        mov     $1020,sp        / initialize stack pointer
        mov     sp,r0
1:
        clr     (r0)+           / clear memory
        br      1b              / and loop

trapd:                          / come here when out of RAM
        sub     $6,r0           / get pointer back to ram
        mov     r0,sp           / set up stack
        mov     $main,r1        / pointer to program
        mov     $1000,r2        / word counter
        sub     $774,r0         / set pointer to relocate program
        sub     r1,r2           / set word counter
        asr     r2
        mov     r0,r4           / remember where program started
1:
        mov     (r1)+,(r0)+     / copy program up
        sob     r2,1b

        clr     (sp)            / set 'return address'
        jmp     (r4)            / and start program
////////////////////////////////////////////////////////////////////////////////

////////////////////////////////////////////////////////////////////////////////
// Actual bootstrap program, position independent code.
//
// Register Assignments
// r0 — Word counter
// r1 — Bus address
// r2 — Number of bytes to transfer for this sector
// r3 — current sector
// r4 — current track
// r5 — timer

main:
        bit     $RX02_MD,*$RXCSR        / is disk configured correctly?
        bne     1f              / yes, skip ahead
        clr     r0              / otherwise set error code
        halt                    / and halt
1:
        mov     $STRACK,r4      / start of system
        mov     $SSECTOR,r3
        mov     $SECSIZW,r2     / read first block
        clr     r1              / into location 0
        jsr     pc,rx_reada
        cmp     $MAGIC,*$0      / is first word the magic number
```

```
        beq     1f              / yes, skip ahead
        mov     $1,r0           / otherwise set error code
        halt                    / and halt
1:
        mov     *$2,r0          / set up counter
        add     *$4,r0
        bcc     1f              / if no overflow, skip ahead
7:
        mov     $4,r0           / otherwise set error code
        halt                    / and halt
1:
        cmp     pc,r0           / will system overwrite this section?
        blo     7b              / yes, go report error
        ror     r0              / set word count (note, carry must be
                                / clear or previous branch would have been
                                / taken)
        mov     $170,r5         / loop counter
        mov     $20,r2          / pointer
copydn:
        mov     (r2)+,(r1)+     / copy word down
        dec     r0              / decrement word count
        beq     loop            / if done, skip ahead
        sob     r5,copydn       / otherwise decrement count and repeat
loop:
        tst     r0              / any bytes to transfer?
        bne     1f              / yes, transfer them
        rts     pc              / no, 'return' (start booted system)
1:
        inc     r3              / increment current sector
        cmp     $LSECTOR,r3     / last sector on track?
        bhis    1f              / no, just continue
        inc     r4              / otherwise increment current track
        mov     $1,r3           / reset sector count
1:
        mov     $SECSIZW,r2     / number of words to transfer
        cmp     r2,r0           /less than one sector left?
        blo     1f              / yes, transfer entire sector
        mov     r0,r2           / otherwise just transfer remaining portion
1:
        sub     r2,r0           / decrement counter
        jsr     pc,rx_read      / read the sector
        add     $SECSIZW * 2,r1 / update bus address
        br      loop            / and start the loop again
////////////////////////////////////////////////////////////////////////////////

////////////////////////////////////////////////////////////////////////////////
// rx_read -- read sector

rx_read:
        jsr     pc,rx_wait      / wait for completion
rx_reada:
        mov     $HI_DENS | READ | GO, *$RXCSR   / send read command
        jsr     pc,rx_pwait     / wait for transfer request flag
        mov     r3,*$RXDBUF     / send sector number
        jsr     pc,rx_pwait     / wait for transfer request flag
        mov     r4,*$RXDBUF     / send track number
```

```
        jsr     pc,rx_wait      / wait for completion
        mov     $HI_DENS | EMPTY | GO, *$RXCSR  / send empty buffer command
        jsr     pc,rx_pwait     / wait for transfer request flag
        mov     r2,*$RXDBUF     / send word count
        jsr     pc,rx_pwait     / wait for transfer request flag
        mov     r1,*$RXDBUF     / send bus address
        jsr     pc,rx_wait      / wait for completion
        rts     pc              / then return
////////////////////////////////////////////////////////////////////////

////////////////////////////////////////////////////////////////////////
// rx_wait -- wait for completion of command, check error status

rx_wait:
1:
        bit     $RX_DONE,*$RXCSR        / done flag set?
        beq     1b              / no, just wait
        tst     *$RXCSR         / error bit set?
        bpl     1f              / no, skip ahead
7:
        mov     $3,r0           / otherwise set error code
        halt                    / and halt
1:
        bit     $INIT_DONE,*$RXDBUF     / power fail?
        bne     7b                      / yes, go report error
        rts     pc              / otherwise return
////////////////////////////////////////////////////////////////////////

////////////////////////////////////////////////////////////////////////
// rx_pwait -- wait for transfer request flag

rx_pwait:
        mov     $30000.,r5      / set up timeout check
1:
        dec     r5              / decrement timer
        bne     2f              / not timed out, skip.
        mov     $2,r0           / otherwise set error code
        halt                    / and halt
2:
        tstb    *$RXCSR         / flag up?
        bpl     1b              / no, loop
        rts     pc              / otherwise return
////////////////////////////////////////////////////////////////////////
```

8748 Program Listing

```
;
;
;
;
; 8748 satellite clock interface
;
; W. Eric Norum
; University of Saskatchewan
;
; This program allows the 8748 to be used as the interface between a
; satellite controlled clock, and an LSI-11.
;
; The internal timer is used to create 400 microsecond 'ticks' which are
; counted and passed to the LSI-11 on demand.  The base time is set by
; the satellite clock if present, or by the LSI-11 itself.
;
;*****************************************************************************
;
; 8748 Resource Allocation
;
; P0    - not used
; P1    - flag indicating that 'cold storage' is full
;
; T0    - flag indicating that the LSI-11 is not ready to recieve data
; T1    - flag indicating that base time data is present and readable
;
; INT   - interrupt request input....indicates that base time data in
;               the 'warming oven' is accurate and should become
;               the current time
; RD    - strobe output to clear T1 flag
; WR    - strobe output to set LSI-11 data ready flag
;
; BUS   - time data to LSI-11
; P1    - set time data from clock (or from LSI-11)
; P20-2 - time data identity to LSI-11
; P23   - flag indicating that the LSI-11 wants the current time saved
;               in 'cold storage'
; P24-6 - time data identity from clock (or LSI-11)
; P27   - interrupt acknoledge line
;
; *****************************************************************************
;
;
; Register allocation
;
; r0    - pointer to remove data from cold storage
; r1    - pointer to place data into warming oven
; r2    - identity of next set time element
; r3    - identity of time element written to the LSI-11
; r4    - number of items remaining to be transferred to the LSI-11
; r5    - temporary storage for accumulator during interrupt routines
;
; r0'   - pointer used during interrupt service routines
; r1'   - pointer used during interrupt service routines
; r2'   - cleared
```

```
; r3'   - contains 1
;
;
; ram layout
;
        .decm

cldstr  .equ    32      ; cold storage (10 bytes)
wrmovn  .equ    42      ; warming oven (8 bytes)
active  .equ    50      ; current time (10 bytes)
;
seconds .equ    0       ; offset to get seconds
minutes .equ    1       ; offset to get minutes
hours   .equ    2       ; offset to get hours
lodays  .equ    3       ; offset to get lo order days
hidays  .equ    4       ; offset to get hi order days
off0    .equ    5       ; offset to get lo order tick count
off1    .equ    6
off2    .equ    7
off3    .equ    8
off4    .equ    9       ; offset to get hi order tick count
;
;
; Input data identity
;
        .hex
isecid  .equ    -20     ; identity bits for seconds
iminid  .equ    -30     ; identity bits for minutes
ihrid   .equ    -40     ; identity bits for hours
ildyid  .equ    -50     ; identity bits for lo order days
ihdyid  .equ    -60     ; identity bits for hi order days
;
imskid  .equ    70      ; mask for identity bits
iincid  .equ    -10     ; increment for identity bits
;
; Output data identity
;
        .oct
osecid  .equ    370     ; seconds
ominid  .equ    371     ; minutes
ohrid   .equ    372     ; hours
oldyid  .equ    373     ; lo days
ohdyid  .equ    374     ; hi days
oo0id   .equ    375     ; lo order offset
oo1id   .equ    376
oo2id   .equ    377
oo3id   .equ    370
oo4id   .equ    371     ; hi order offset
nitems  .equ    10d     ; number of items to send to the LSI-11
;
;
intrvl  .equ    -5      ; number of timer ticks per timer interrupt
;
;
;*****************************************************************************
;
```

```
; Low memory, reset and interrupt vectors
        .oct
        .org    0000
        jmp     reset           ; go to power-on reset routine

        .org    0003
        jmp     timacc          ; external interrupt (time accurate)

;**********************************************************************
;
; Internal Timer Interrupt Service Routine
;
;**********************************************************************
        .org    0007
        mov     r5,a            ; save accumulator
        mov     a,#intrvl       ; set up internal timer again
        mov     t,a
        sel     rb1             ; switch to alternate registers

        in      a,p2            ; are we to freeze time?
        jb3     freeze          ; yes, go freeze it.
        jmp     incoff          ; otherwise just increment time offset

freeze                          ; freeze.....copy time to cold storage
        mov     r0,#active      ; point r0 to current time
        mov     r1,#cldstr      ; point r1 to cold storage

        mov     a,@r0           ; get byte of current time
        mov     @r1,a           ; and save it cold storage
        inc     r0              ; increment current time pointer
        inc     r1              ; increment frozen time pointer

        mov     a,@r0           ; get byte of current time
        mov     @r1,a           ; and save it cold storage
        inc     r0              ; increment current time pointer
        inc     r1              ; increment frozen time pointer

        mov     a,@r0           ; get byte of current time
        mov     @r1,a           ; and save it cold storage
        inc     r0              ; increment current time pointer
        inc     r1              ; increment frozen time pointer

        mov     a,@r0           ; get byte of current time
        mov     @r1,a           ; and save it cold storage
        inc     r0              ; increment current time pointer
        inc     r1              ; increment frozen time pointer

        mov     a,@r0           ; get byte of current time
        mov     @r1,a           ; and save it cold storage
        inc     r0              ; increment current time pointer
        inc     r1              ; increment frozen time pointer

        mov     a,@r0           ; get byte of current time
        mov     @r1,a           ; and save it cold storage
        inc     r0              ; increment current time pointer
        inc     r1              ; increment frozen time pointer
```

```
        mov     a,@r0           ; get byte of current time
        mov     @r1,a           ; and save it cold storage
        inc     r0              ; increment current time pointer
        inc     r1              ; increment frozen time pointer

        mov     a,@r0           ; get byte of current time
        mov     @r1,a           ; and save it cold storage
        inc     r0              ; increment current time pointer
        inc     r1              ; increment frozen time pointer

        mov     a,@r0           ; get byte of current time
        mov     @r1,a           ; and save it cold storage
        inc     r0              ; increment current time pointer
        inc     r1              ; increment frozen time pointer

        mov     a,@r0           ; get byte of current time
        mov     @r1,a           ; and save it cold storage
                                ; Time now frozen...

        sel     rb0             ; select original registers
        mov     r0,#cldstr      ; set output pointer to frozen time
        mov     r3,#osecid      ; set output identity to seconds
        mov     r4,#nitems      ; set output counter to number of bytes
        sel     rb1             ; and select alternate registers again.

        jf1     incoff          ; if flag is up, leave it up
        cpl     f1              ; if down, put it up.

incoff                          ; incoff....increment time offset
        mov     r0,#active+off0 ; set pointer to low order offset

        mov     a,@r0           ; get first byte of offset
        add     a,r3            ; increment byte
        mov     @r0,a           ; save updated byte

        inc     r0              ; increment pointer
        mov     a,@r0           ; get next byte
        addc    a,r2            ; add carry of previous operation to it
        mov     @r0,a           ; then save it

        inc     r0              ; increment pointer
        mov     a,@r0           ; get next byte
        addc    a,r2            ; add carry of previous operation to it
        mov     @r0,a           ; then save it

        inc     r0              ; increment pointer
        mov     a,@r0           ; get next byte
        addc    a,r2            ; add carry of previous operation to it
        mov     @r0,a           ; then save it

        inc     r0              ; increment pointer
        mov     a,@r0           ; get next byte
        addc    a,r2            ; add carry of previous operation to it
        mov     @r0,a           ; then save it
                                ; offset now updated
```

```
        sel     rb0             ; select original registers
        mov     a,r5            ; restore accumulator
        retr                    ; timer interrupt service routine completed
;
;
;*****************************************************************************
;
; External Interrupt Service Routine    (Time Accurate)
;
;*****************************************************************************
;
        .org    0400
timacc
        dis     tcnti           ; turn off timer interrupts
        mov     r5,a            ; save accumulator
        mov     a,#intrvl       ; reset timer
        mov     t,a

        anl     p2,#177         ; acknowledge the interrupt
        orl     p2,#200
        ins     a,bus           ; clear the data strobe bit

        sel     rb1             ; select alternate registers

        mov     r0,#wrmovn      ; point r0 to time in warming oven
        mov     r1,#active      ; point r1 to active time storage

        mov     a,@r0           ; get byte of warming oven time
        mov     @r1,a           ; and save it in the active time area
        inc     r0              ; increment pointers
        inc     r1

        mov     a,@r0           ; get byte of warming oven time
        mov     @r1,a           ; and save it in the active time area
        inc     r0              ; increment pointers
        inc     r1

        mov     a,@r0           ; get byte of warming oven time
        mov     @r1,a           ; and save it in the active time area
        inc     r0              ; increment pointers
        inc     r1

        mov     a,@r0           ; get byte of warming oven time
        mov     @r1,a           ; and save it in the active time area
        inc     r0              ; increment pointers
        inc     r1

        mov     a,@r0           ; get byte of warming oven time
        mov     @r1,a           ; and save it in the active time area
        inc     r0              ; increment pointers
        inc     r1
                                ; now clear the time offset

        clr     a
        mov     @r1,a           ; clear byte of offset
        inc     r1              ; and update pointer
```

```
        mov     @r1,a           ; clear byte of offset
        inc     r1              ; and update pointer

        mov     @r1,a           ; clear byte of offset
        inc     r1              ; and update pointer

        mov     @r1,a           ; clear byte of offset
        inc     r1              ; and update pointer

        mov     @r1,a           ; clear byte of offset

        sel     rb0             ; select original registers

        mov     r2,#isecid      ; set input id to seconds
        mov     r1,#wrmovn      ; set input pointer to warming oven

        mov     a,r5            ; restore accumulator
        en      tcnti           ; enable timer interrupts again

        retr                    ; and return
;
;
;
;*****************************************************************************
;
; Power Up Routine and Main Program
;
;*****************************************************************************
;
        .org    1000
reset                           ; reset.....entered on power up.
;
; Clear storage
;
        mov     r0,#cldstr      ; set pointer to start of storage
        mov     r1,#active+10d-cldstr ; set up loop counter
        clr     a               ; clear accumulator
clrstr
        mov     @r0,a           ; clear byte
        inc     r0              ; update pointer
        djnz    r1,clrstr       ; and loop till done
;
; set up registers
;
        mov     r1,#wrmovn      ; set input pointer to warming oven
        mov     r2,#isecid      ; set input id to seconds
        sel     rb1             ; select alternate registers
        mov     r2,#0           ; put 0 in r2'
        mov     r3,#1           ; put 1 in r3'
        sel     rb0             ; select original registers again

        mov     a,#intrvl       ; set time invterval
        mov     t,a

        en      tcnti           ; enable timer interrupts
```

```
        en      i               ; enable external interrupts
        strt    t               ; start timer
;
; Main routine.....input or output data when able
;
loop
        jf1     full            ; if flag is up, cold storage is full
        jmp     empty           ; otherwise it is empty
full
        jt0     empty           ; if LSI-11 not ready, skip ahead

        mov     a,r3            ; get identity
        outl    p2,a            ; and send it
        inc     r3              ; increment identity
        inc     a               ; check to see if it has wrapped around
        jnz     okay            ; if not, skip ahead
        mov     r3,#oo3id       ; otherwise fix it up
okay
        mov     a,@r0           ; get data byte
        outl    bus,a           ; and send it to the LSI-11
        inc     r0              ; increment pointer
        djnz    r4,empty        ; all data sent?
        clr     f1              ; yes, pull down flag

empty
        jnt1    loop            ; if no time data, just loop
        ins     a,bus           ; otherwise clear the flag
        in      a,p2            ; read the identity
        anl     a,#imskid       ; mask for input identity
        add     a,r2            ; compare with desired identity
        jnz     loop            ; if not correct, loop

        in      a,p1            ; otherwise read the data
        mov     @r1,a           ; store the data
        inc     r1              ; increment the pointer

        mov     a,r2            ; get identity
        add     a,#iincid       ; set it to the next item
        mov     r2,a            ; and put it back

        jmp     loop            ; then repeat

        .end
```