PB88-232582

Publication No. FHWA/RD-87/044
December 1987

Reference Manual for the UMTRI/FHWA Road Profiling (PRORUT) System

U.S. Department of Transportation
Federal Highway Administration

Research, Development, and Technology
Turner-Fairbank Highway Research Center
6300 Georgetown Pike, McLean, Va 22101-2296

FOREWORD

Two methods are available for measuring road roughness for pavement condition surveys. Both can be used at highway speeds without interfering with traffic. One method uses Response Type Road Roughness Measuring (RTRRM) systems. This method measures the response of an instrumented car or trailer to road roughness. The response depends on the vehicle, its condition, and the speed of measurement. The second method measures the roadway profile, independent of the vehicle and operating conditions. RTRRM systems are widely used because the equipment is inexpensive. However, RTRRM systems require frequent calibrations, and the measurement depends on many factors difficult to control.

Road roughness profiling is preferable to response type measurements. The roughness profile can be obtained with sufficient accuracy and reliability. The recorded profile can be used for calculating rideability, for calculating change in PSI (Present Serviceability Index) over time, for calculating the amount of overlay needed for resurfacing, for calibrating RTRRM systems, and more.

In recent years, reliable non-contact height sensors have become available, making profiling equipment attractive. Furthermore, the same type of sensor can be used for measuring rut depth. The PRORUT system developed for FWHA by the University of Michigan provides an average rut depth by adding one height sensor centered between the two sensors measuring the wheel tracks. The signal processing and data analysis for profiles and rut depth are integrated. This three-sensor system can be expanded by adding sensors to increase the accuracy in measuring rut depth. However, the required accuracy depends on the use of the data, and for most applications the average rut depth is probably adequate.

Donald G. Fohs

Director, Office of Engineering
and Highway Operations
Research and Development

NOTICE

Technical Report Documentation Page

1. Report No.	2. Government Accession No.	3. Recipient's Catalog No.
FHWA/RD-87/044		PB88 232582/AS
4. Title and Subtitle REFERENCE MANUAL FOR THE UMTRI/FHWA ROAD PROFILING (PRORUT) SYSTEM		5. Report Date December 1987
		6. Performing Organization Code
7. Author(s) M. R. Hagan and M.W. Sayers		8. Performing Organization Report No. UMTRI-87-5
9. Performing Organization Name and Address The University of Michigan Transportation Research Institute 2901 Baxter Road, Ann Arbor, Michigan 48109		10. Work Unit No. (TRAIS) 31W3-062
		11. Contract or Grant No. DTFH61-83-C-00123
12. Sponsoring Agency Name and Address Federal Highway Administration U.S. Department of Transportation Washington, D. C. 20590		13. Type of Report and Period Covered Final 9/83 - 1/87
		14. Sponsoring Agency Code

15. Supplementary Notes

Contract Technical Representative: Dr. R.R. Hegmon

16. Abstract

The objectives of this project were to assess the capabilities that are needed in a road profilometer and develop a design tailored to minimize life costs of the system. This led to the development of a system based on the IBM PC microcomputer. With the exception of a signal-conditioning unit, the system is constructed from commercial components. The software controls the measurement of road profile and rut depth, the viewing of the data, and daily checks of the hardware integrity. This document is the reference manual for the profilometer—presently known as the PRORUT system. The manual is intended to document the software and hardware components of the system, and is of special interest to the technical staff responsible for maintenance, repair, or modification of the system when required.

There are three companion reports prepared as part of the same project. One gives an overview of the project (FHWA/RD-87/042), another is the PRORUT User's manual (FHWA/RD-87/043), and the third describes the validation of the FHWA profilometer along with other profilometers at a profilometer meeting (FHWA/RD-86/100).

17. Key Words	18. Distribution Statement
longitudinal profile, profilometer, road roughness, quarter-car, digital data acquisition systems, IBM PC, rut depth, road profile	No restrictions. This document is available to the public through the National Technical Information Service, Springfield, VA 22161

19. Security Classif. (of this report)	20. Security Classif. (of this page)	21. No. of Pages	22. Price
Unclassified	Unclassified	306	

TABLE OF CONTENTS

TABLE OF CONTENTS (continued)

LIST OF FIGURES

LIST OF FIGURES (Continued)

Figure *Page*

LIST OF TABLES

INTRODUCTION

This manual describes the technical details of the hardware and software that comprise a profiling and rut depth (PRORUT) measuring system that was designed and built by The University of Michigan Transportation Research Institute (UMTRI) as a part of the FHWA project "Methodology for Road Roughness Profiling and Rut Depth Measurement," Contract No. DTFH61-83-C-00123. The system is is simply called "the profilometer" in this manual, although it is sometimes called the PRORUT in other documents. This manual is part of a series of four reports dealing with the profilometer. The others are:

- *Methodology for Road Roughness Profiling and Rut Depth Measurement*—a summary report for the project under which the profilometer was designed and built.[1]
- *User's Manual for the UMTRI/FHWA Road Profiling (PRORUT) System*—the instruction manual for operating the system.[2]
- *The Ann Arbor Profilometer Meeting*—a report that describes some of the testing and analysis methods used in its development, along with validation results.[3]

The FHWA profilometer is a digital data acquisition system based on an IBM PC computer and 3M cartridge tape drive, with software that aids in the collection, processing, and viewing of the data. It is built on the chassis of a 1974 Dodge B300 van provided by FHWA. In addition to the computer system, the hardware also includes transducers for measuring the speed of travel, vertical accelerations on the vehicle body above each wheeltrack, and height above the road at the midpoint and in each wheeltrack. The electronics system has provisions for two additional outboard road sensors so that rut depth in the separate wheeltracks can be measured should additional sensors be installed.

The following section, "Hardware," describes the hardware components used in the profilometer. Hardware designed at UMTRI is described in this section, schematics are provided in appendix A, and cabling information is included in appendix B. Most of the hardware components are commercially available, and detailed documentation for those components is provided by the manufacturers.

The software for the computer system contains the following commercially available elements:

- IBM DOS 2.1 .
- Microsoft Fortran (subset of Fortran 77).
- Halo Graphics Package (with Fortran drivers).
- UMTRI Fortran Library.

The rest of the software is written in Fortran to perform tasks that are specific to the profilometer. The next four sections of this manual describe this software.

The section "Data Collection" describes the techniques used to control the hardware of the system to collect data measured with the sensors on board the profilometer and record those measures on tape. Next, the section entitled "Computation Methods" describes the numerical methods used to compute longitudinal profile, rut depth, and roughness from the transducer signals. The section "Data Files" describes the structures of the files that are created by the profilometer. The final section in the manual, "Profilometer Subroutines," lists all of the Fortran procedures that were written to perform the tasks described in the preceding sections and in the users manual. Appendix C documents a library of Fortran extensions that allows improved handling of files and the display screen. Appendix D contains the source listings of the profilometer code.

HARDWARE

The FHWA profilometer is a digital data acquisition system, with appropriate transducers, and software that aids in the collection, processing, and viewing of profile and rut depth data. It is built on the chassis of a 1974 Dodge B300 van provided by FHWA. In addition to the computer system, the hardware also includes transducers for measuring the speed of travel, height above the road at the midpoint and in each wheeltrack, and vertical accelerations on the vehicle body at two of the height sensors. The electronics system has provisions for two additional outboard road sensors so that rut depth in the separate wheeltracks can be measured should additional sensors be installed.

This section describes the hardware components used in the profilometer. The computer components and transducers, described briefly in this section, are commercially available products that include detailed documentation provided by the manufacturers. For this reason, only the UMTRI-supplied hardware (analog signal-conditioning unit and calibration IBM interface card) is described in detail. Appendix A contains all of the schematics for the hardware built at UMTRI, and appendix B contains all of the cabling information.

The Computer System

An IBM PC is the heart of the profilometer serving to control its calibration, operation, data acquisition, data processing, and data viewing. An ADIC Model 550 Tape Recorder system is used for recording the measured data. It is a 64-megabyte cartridge recorder capable of recording at up to 35k Hz, which appears as four hard disk drives to the computer. The PC is a standard commercial version with the following components:

- IBM-PC, 256k memory, 2 DS DD floppy disk drives, and floating point processor.
- Hercules graphics card and IBM monochrome monitor.
- AST Six Pack Plus with 384k memory, clock, serial port, and parallel port.
- Data Translation Analog I/O, Model DT-2801-A.
- Hicomp 512k bubble memory card.
- C. Itoh dot matrix printer (IBM compatible).
- ADIC tape control card.
- Calibration IBM interface card (custom-built with the signal-conditioning unit).

A Tecmar expansion chassis is required with the PC to accommodate all of the extra cards. Figure 1 shows an overview of these components.

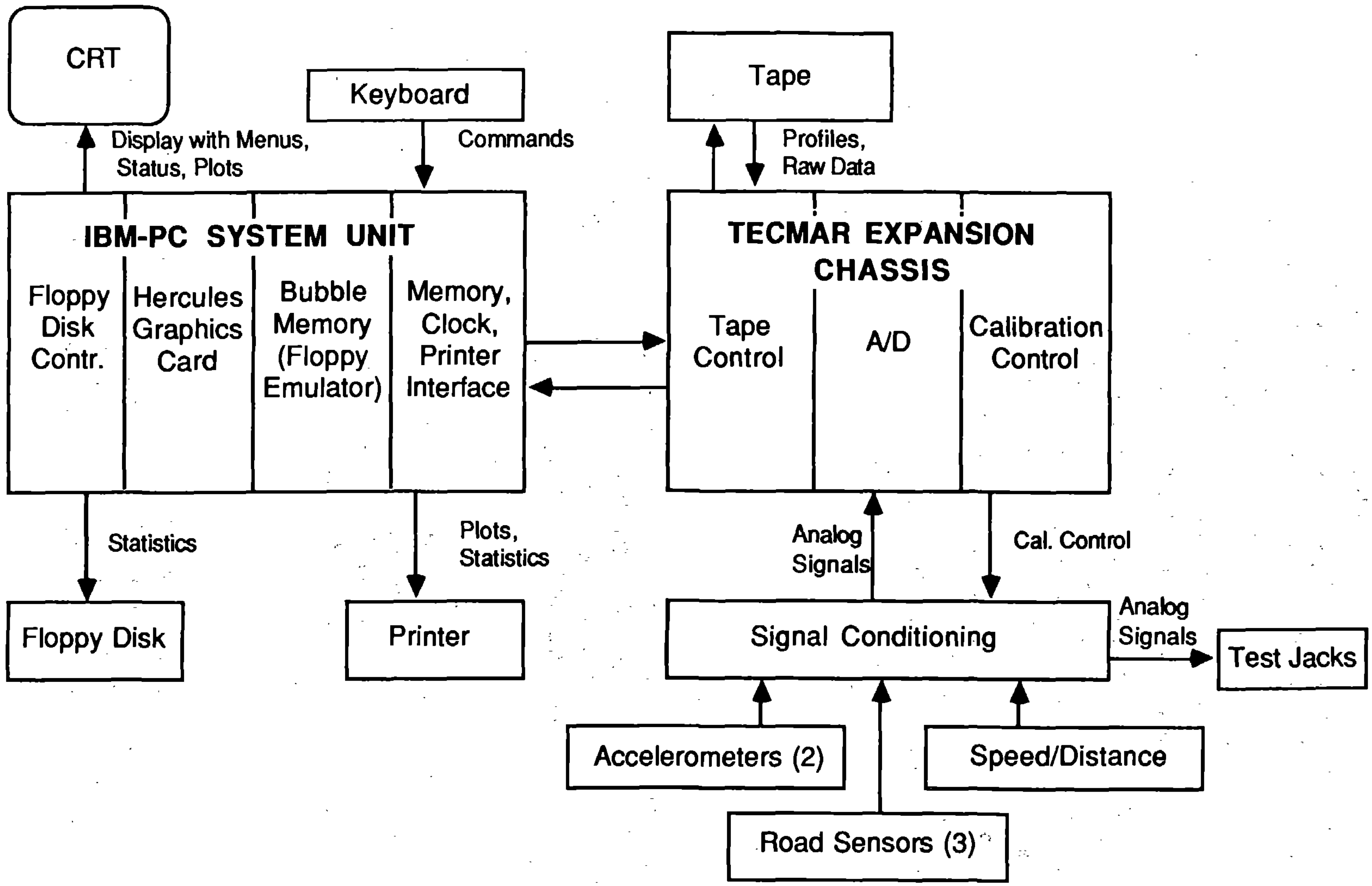


Figure 1. Overview of the hardware used in the profilometer.

The system software is installed on the 512k bubble memory card, which is retained in memory even when the system power is turned off. The floppy disks are not necessary for operation of the system, but can be used to transport information into and out of the computer. The AST Six Pack Plus supplements the computer memory to the limit of 640k, and the parallel port is used to drive the printer. The Data Translation Analog I/O is the analog-to-digital converter system. It has 8 double-ended inputs and is capable of digitization at 35k Hz. The ADIC Tape Control card is purchased from the tape manufacturer, and is used to control the tape recording system.

The conditioning of the transducer signals is performed by an *UMTRI analog signal conditioning unit*, described in a later subsection. The calibration IBM interface card is a custom-built card designed to interface the IBM PC with the UMTRI analog signal-conditioning unit, communicating signals needed for calibrating the data channels.

The IBM PC, the expansion chassis, and the tape recorder are mounted in the instrumentation console behind the driver's seat, which is shock mounted with cable isolators purchased from Aeroflex. The printer is mounted atop the console, and the keyboard and monitor are installed on a pedestal next to the front passenger seat. (Photographs of the system are contained in the users manual and the project report.[1,2])

Transducers

Speed/Distance Transducer

The vehicle speed and distance of travel are measured by a pulser installed in the left front wheel. Within the back side of the disc brake rotor is installed an exciter ring in the form of a disc with 120 notches (3 degree intervals). Rotation is sensed by a magnetic pickup which generates two pulses with the passage of each notch. Each distance pulse corresponds to approximately 0.37 in (10 mm) of forward travel for the installed tires. (The exact relationship between the pulse interval and forward travel is determined by calibration involving a measured distance. See section 3.3.3 in the User's Manual.[2]) The pulse train goes to the analog signal-conditioning unit, where it feeds into a frequency-to-voltage converter to produce an analog speed signal. The pulse train is also fed to the computer to communicate distance traveled. Within the computer, a counter synchronized to the pulse train triggers data sampling at the selected intervals along the road, as described in the next section, "Data Collection."

Accelerometers

The accelerometers are rigidly mounted in the vertical orientation on fixtures just above the road sensors in each wheeltrack. These are Sunstrand Model QA-900 servo accelerometers, rated at 30 g's full range (250 g's shock), and 500 Hz natural frequency. They have a threshold and resolution each better than 0.005 mg, and a maximum cross-axis sensitivity of 2 mg/g. They are powered by ±15 volts DC supplied from the signal-conditioning system.

Road Height Sensors

Two types of road sensors were provided for test by FHWA— infrared noncontacting sensors developed by Southwest Research Institute, and a set of Selcom Optocators. [4,5,6] The van is modified by installation of enclosures below the floor level where the sensors are mounted. This is necessary primarily to place the sensors at the proper distance from the road, nominally 10 in (250 mm), but has the additional advantage of minimizing obstruction of the vehicle interior.

The enclosures are designed to accommodate either sensor. The infrared sensors are self-contained, requiring only a 12 volt DC power supply (obtained directly from the inverter connections) and signal wires going to the signal conditioner. The Selcom Optocators require their own signal conditioning box, which is mounted at the rear of the van.

Analog Signal-Conditioning Unit

Backplane

A picture of the signal conditioning unit backplane is shown in figure 2. The backplane is a printed circuit card that holds the connectors for the transducer inputs, signal conditioning cards, control inputs, control card, and pull-up card. Also on the backplane are solder pads for the A/D and test jack wiring. Most of the connections between cards are accomplished with printed circuit traces, although some wire wrapping is used. Figure 3 shows the layout of the backplane. Transducers are interfaced via the 9-pin connectors at the top. Two 25-pin connectors (DB25/A and DB25/B) bring in the calibration control lines from the computer. The control card in the far right slot (as shown in the figure) decodes these signals and distributes them to the remaining cards. The next 16 slots are for signal conditioning cards. In the profilometer system, slots C0 through C7 are occupied by analog signal conditioning cards, C8 by the velocity converter card, and C15 by the A/D check card. Slots C9 to C14 are not used for the profilometer. The last slot contains the pull-up card. Power is brought in via a screw terminal on the left end of the backplane.

Test Jacks

Test jacks for monitoring the analog signals that get fed to the A/D converter are mounted on the front panel and are provided for setup and diagnostic purposes. The jacks are mounted in pairs with a ground jack for each signal jack so that standard dual banana plugs may be used.

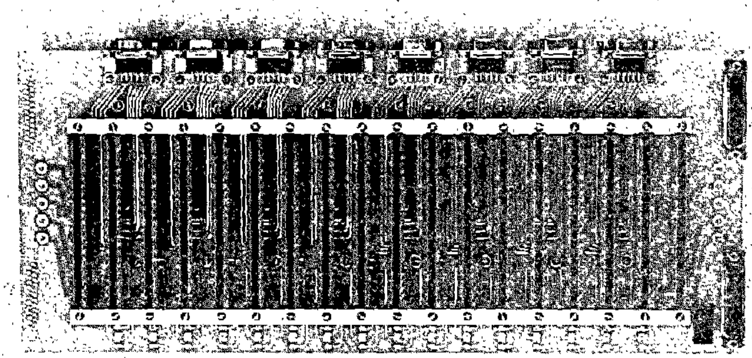


Figure 2. Photograph of the signal-conditioning unit backplane.

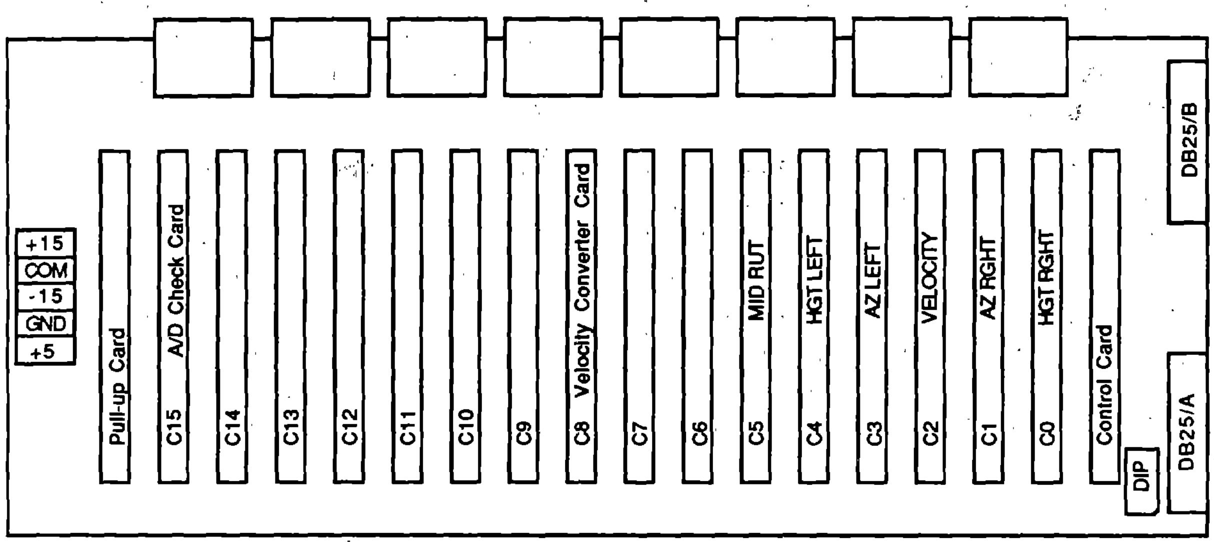


Figure 3. Layout of the signal conditioning unit backplane.

Analog Signal-Conditioning Card

A picture of an analog signal conditioning card is shown in figure 4 and a block diagram is shown in figure 5. The transducer connects to the card via a 9-pin connector (through the backplane) and an I/O header. The jumpers on this header provide the transducer's excitation (±15 volts for accelerometers and rate transducers and 0-10 volts for strain gauges and potentiometers) and route the transducer outputs through the calibration relay to the instrumentation amplifier. The computer measures the gain of the card by switching this relay. The inputs to the instrumentation amp from the transducer are disconnected and a D/A-generated calibration signal (staircase waveform) is inserted.

The output of the card is measured by the A/D and the amplifier gain is calculated using a linear regression formula. (The same hardware can also be used for a strain gauge bridge, in which case the shunt cal relay is used to connect a resistor in parallel with one arm of the bridge. Because the voltage this generates is known to be equivalent to some force, the computer can calculate the overall system gain.)

The instrumentation amp, excitation regulator, and buffer amp in the dashed rectangle of the block diagram reside in an Analog Devices 2B31 module. A 16-pin gain header provides the connection of an offset pot to the instrumentation amp, the gain setting resistor, the shunt cal resistor, and jumpering of the output of the instrumentation amp to the input of the buffer amp. The computer adjusts the offset voltage of the card by sending a digital value to an 8-bit D/A whose output is summed with the signal at the buffer amplifier. Finally, the signal is filtered by a 4-pole Butterworth filter (contained on a plug-in card) whose cutoff frequency is proportional to a clock frequency that the computer generates.

Filter Card

The filter card plugs onto the analog signal conditioning card. It contains a ±6 volt regulator, a MF10 filter chip, and a filter for removing clock feedthrough on the output of the filter. The filter is composed of two stages cascaded to give a 4-pole Butterworth filter. Three resistors for each stage set the gain and the Q of the filter. The cutoff frequency of the filter is proportional to the clock frequency delivered to the filter by the backplane. For more information on the MF10 filter chip see a National Semiconductor Data book.

Configuring a Channel

A channel is configured for a specific transducer by entering the needed information into the setup table of the profilometer software and also by putting the appropriate jumpers and resistors on the various headers. Figure 6 shows an example setup table for the profilometer as it is displayed on the display screen when running the profilometer software. (See the User's Manual[2] for details on getting started with the profilometer software.)

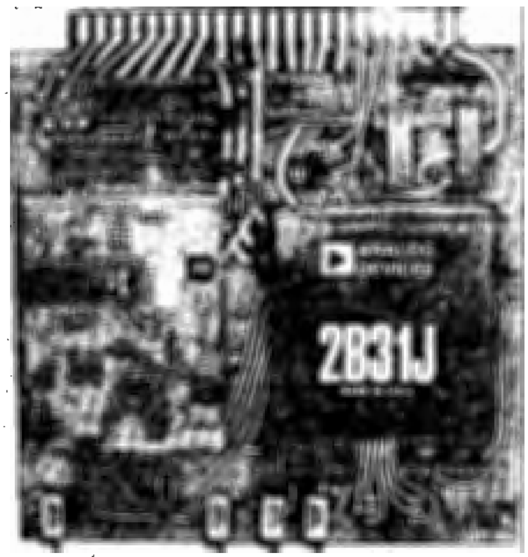


Figure 4. Photograph of an analog signal-conditioning card.

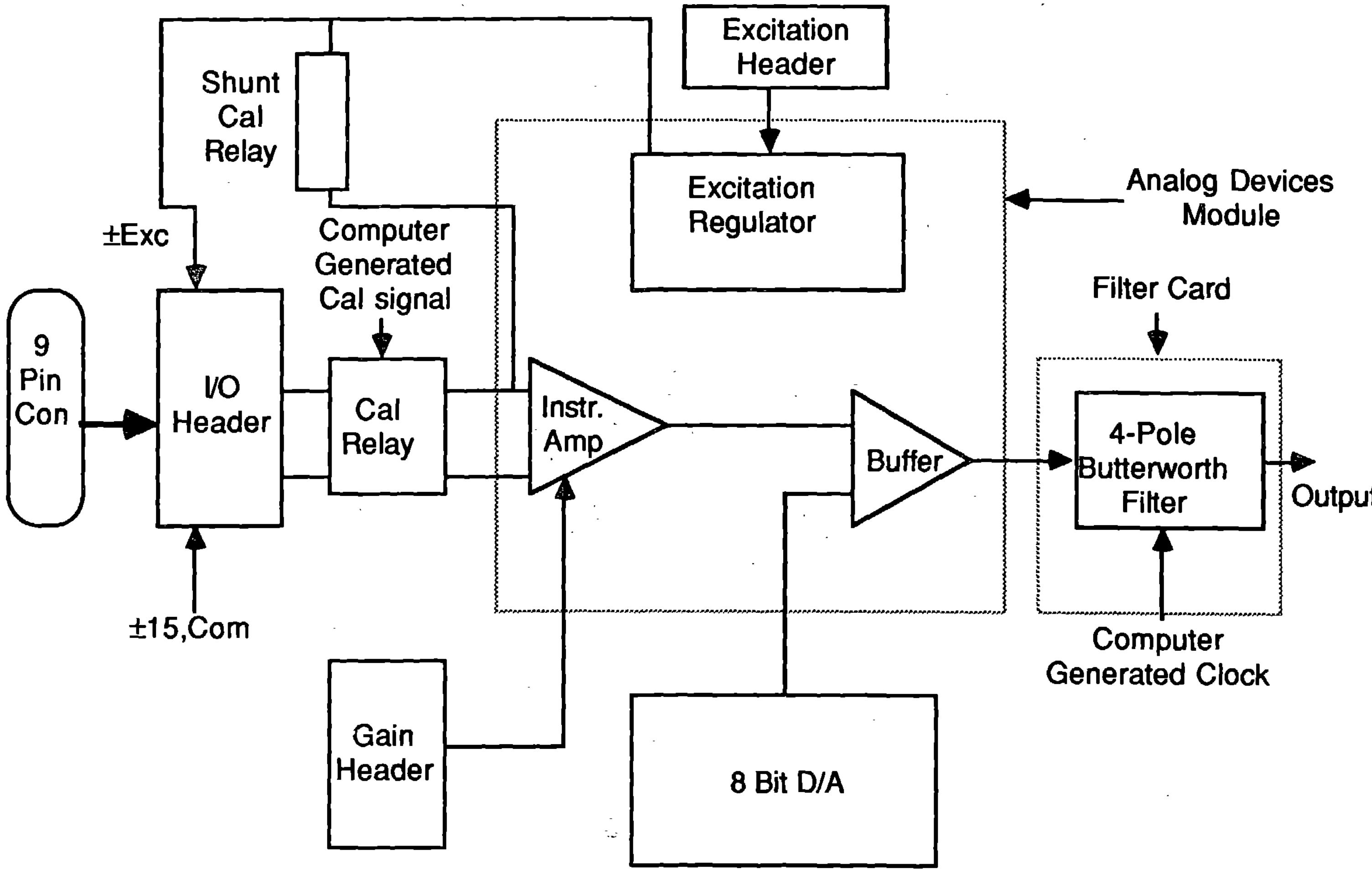


Figure 5. Block diagram for an analog signal-conditioning card.

CHAN #	ID	UNITS	TYPE	TRANSDUCER GAIN	AMPLIFIER GAIN (NOM)	OFFSET AT ZERO VOLTS	AMPLIFIER GAIN (ACT)	FULL SCALE
****	********	********	****	**********	*********	**********	*********	******
0	HGT RGHT	INCH	0	1.03950	2.0000	0.0000	2.0013	2.5970
1	AZ RGHT	G'S	0	.38760	1.3000	0.0000	1.2980	1.4930
2	VELOCITY	MPH	0	20.73190	1.6000	0.0000	1.6014	64.7312
3	AZ LEFT	G'S	0	.39429	1.3000	0.0000	1.3147	1.4995
4	HGT LEFT	INCH	0	1.04670	2.0000	0.0000	2.0185	2.5928
5	MID RUT	INCH	0	1.05000	2.0000	0.0000	.0000	.0000
6	LEFT RUT	INCH	0	1.04000	2.0000	0.0000	.0000	.0000
7	RGHT RUT	INCH	0	1.04000	2.0000	0.0000	.0000	.0000
8	DISTANCE	INCH	0	.36393	1.0000	0.0000	.0000	.0000

Figure 6. Example display showing setup data for the analog signal-conditioning cards.

The first item entered is the channel ID. This is the name of the channel and can be any string of up to eight characters. The second entry is the type of units associated with the transducer which is also limited to eight characters. The third entry is the transducer type which can be 0, 1, or 2. A transducer of type *0* is one whose zero data value is assumed to correspond to zero volts. In the profilometer, all transducers are type *0* and have their zero data when the calibration bar is in the middle position. When a calibration of a channel of type *0* is performed, the transducer remains connected to the amplifier during the nulling process. The computer assumes that the transducer is in a zero data condition (at rest for an accelerometer) and adjusts the amplifier so that the output is nominally zero volts (±.040 V). The amplifier gain is then measured by the staircase procedure.

(A transducer of type *1* does not have a convenient zero data condition. This often occurs when a pot is used to measure a deflection that never has exactly the same zero position, such as the static deflection of a vehicle suspension. When a calibration of a channel of type *1* is done, the computer disconnects the transducer from the amplifier and shorts the amplifier inputs. Thus only the amplifier is nulled. The amplifier gain is measured by the staircase procedure.)

(A type *2* transducer is a resistive bridge transducer for which the zero procedure is the same as for a type *0* transducer. The gain is measured via a shunt cal resistor.)

The fourth item in the display is the transducer gain (units/volt) for types *0* and *1* or the shunt cal value (in units) for type *2* transducers. For the transducers provided with the profilometer, these gains are measured according to the manufacturer's instructions. The usual calibration method involves providing various levels of input that are known with greater accuracy than will ever be required for the transducer, and measuring the corresponding voltage outputs. At UMTRI, accelerometers are usually calibrated by placing them on a tilt table so that the input is the sine of the tilt angle times gravity. Height sensors are calibrated by attaching them to a machinist's mill and moving the bed of the mill to provide the input displacement.

The fifth item shown on the display is the nominal amplifier gain, with units of volt/volt. This should be close (±10%) to the actual gain because the calibration algorithm uses it to calculate the input step size for the staircase waveform.

The next item is an offset, which is defined as zero for all of the transducers used in the profilometer.

The actual amplifier gains cannot be changed by editing the screen. It is measured automatically during an electical calibration, as described in section 3.1 in the User's Manual. The value shown is the result obtained from the most recent calibration.

The final item is the full-scale value, which corresponds to the maximum reading of 5 volts. It is calculated for a type *0* or *1* transducer with the relationship:

Full scale = [transducer gain/amp gain] $\times$ 5

For example, in the example setup from figure 6, the full scale for channel 0 is

Full scale = 1.0395 (inches/v) / 2.0013 (v/v) × 5 (v) = 2.59706 inches

and for channel 2 it is

Full scale = 20.7319 (mi/h/v) / 1.6014 (v/v) × 5 (v) = 64.73 mi/h

Usually, the desired full scale is used to determine an appropriate amplifier gain.

Analog Control Card

The analog control card occupies the far right slot on the backplane (see figure 3). The address and strobe lines from the DB connectors are routed to this card. Figure 7 shows a block diagram of this card. One decoder decodes the address lines and the calibration relay enable line into the 16 different relay control signals. One of these signals goes to each analog signal-conditioning card and either turns on or turns off the calibration relay. The second decoder decodes the address lines and D/A enable line into 16 different D/A enable lines. These lines enable the 8-bit data bus to be loaded into a offset D/A on the selected analog signal-conditioning card.

Velocity-Converter Card

The velocity-converter card occupies slot C8 on the backplane. Figure 8 depicts a functional diagram of this card. The signal from the magnetic pickup on the left front wheel comes in the 9-pin connector on the backplane and is fed into a LM2917 frequency-to-voltage converter chip. This chip uses a frequency doubler so that the resolution of the pulser is effectively multiplied by two. The output from this device is an analog signal proportional to velocity, which then goes to a 2-pole filter that removes the ripple. The filtered velocity signal is then directed to the velocity analog signal-conditioning card elsewhere on the backplane.

The original pulser output also goes to a pulse shaper and a differential line driver whose output is a digital signal that is twice the frequency of the wheel pulser. This signal proceeds to the A/D sequencer where it synchronizes the data sampling to a multiple of wheel pulses (fixed distance sampling).

A/D Check Card

The A/D check card occupies slot C15 on the backplane. Figure 9 diagrams the operation of this card. The output of this card goes to the channel 7 input on the A/D card. Normally the output of the analog signal-conditioning card in slot C7 is connected to the A/D. For test purposes, this signal is removed and either a 2.5 volt reference or the calibration D/A signal is routed to the A/D. The 2.5 volt reference allows the gain of the A/D converter to be checked. The gain calibration of the D/A can be checked when it is connected to the A/D.

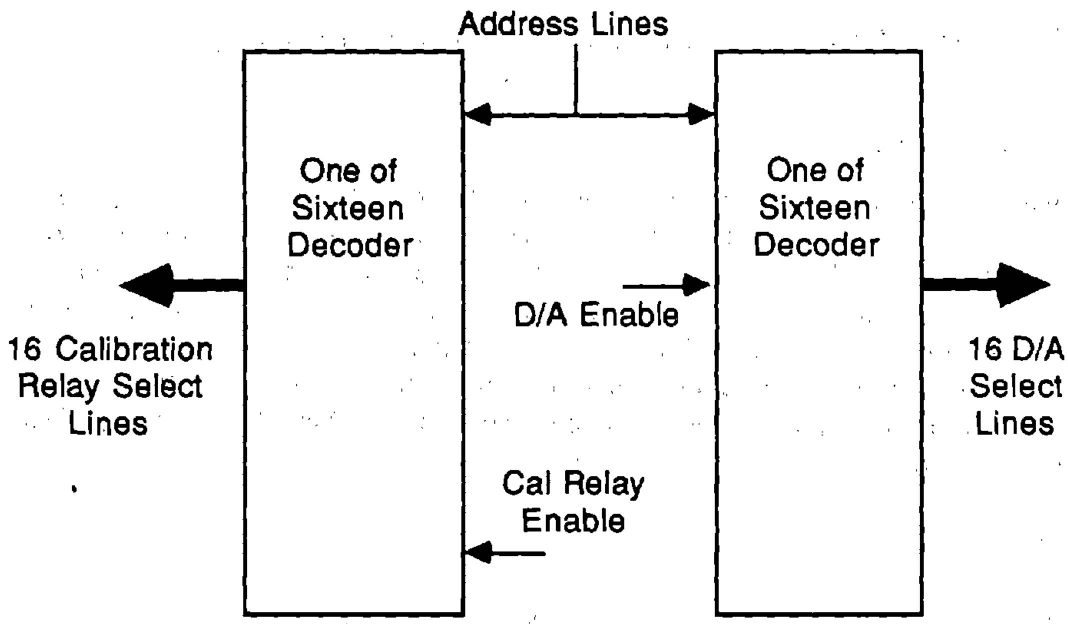


Figure 7. Block diagram of the analog control card.

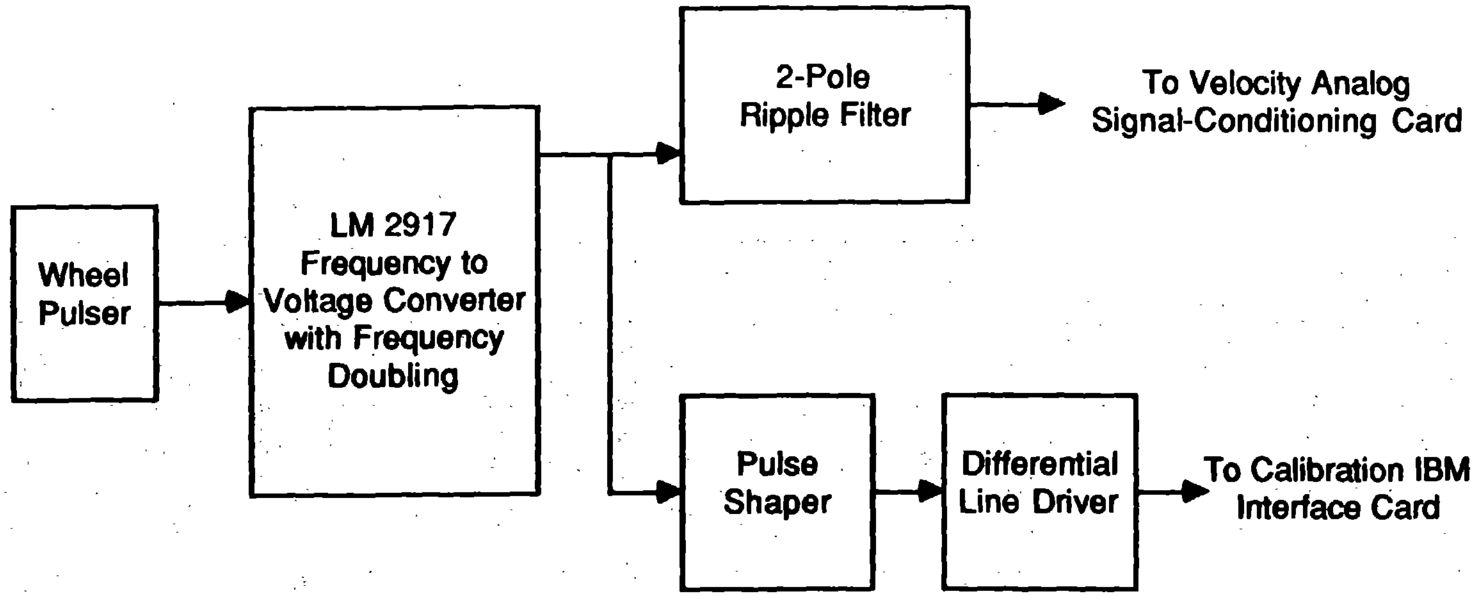


Figure 8. Block diagram of the velocity converter card.

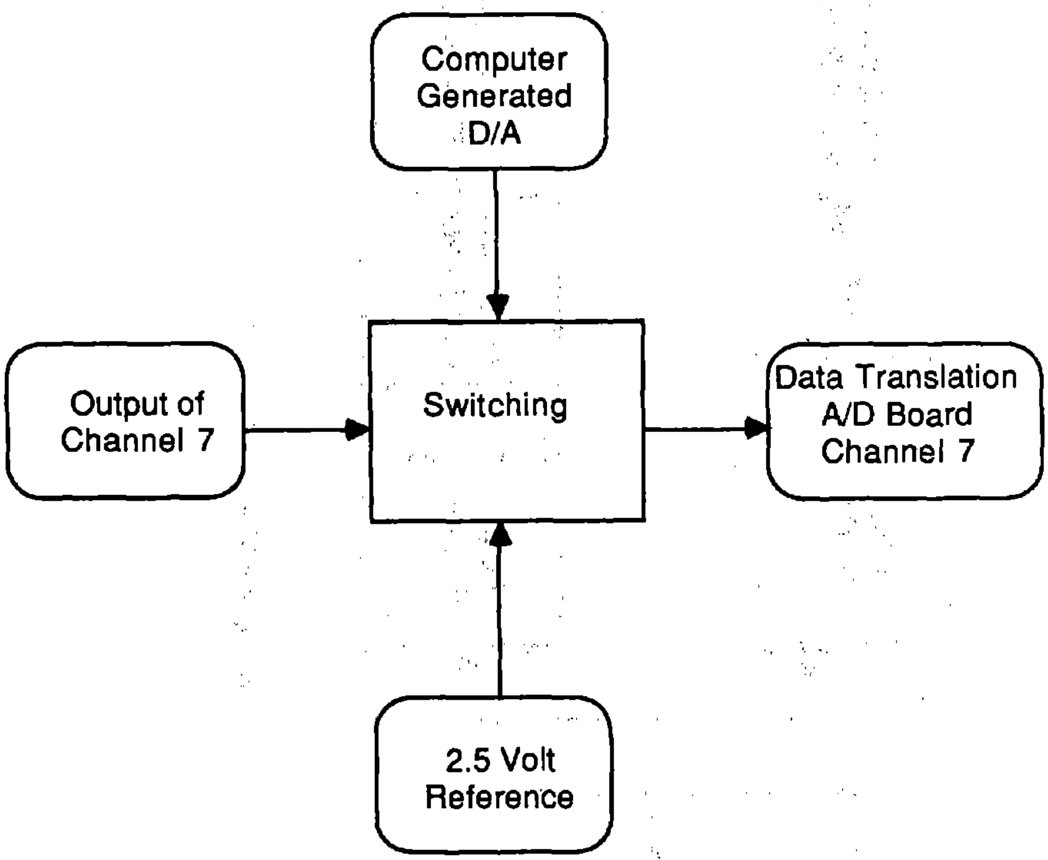


Figure 9. Functional diagram of the A/D check card.

Pull-up Card

The pull-up card occupies the left most slot on the backplane. This card contains a voltage reference that all of the offset D/A's use. Since all of the data bus lines and the filter clock come from opto isolators on the calibration IBM interface card, pull-up resistors are required. These resistors reside on this card.

Calibration IBM Interface Card

The calibration IBM interface card plugs into either the IBM PC or the expansion chassis. It is a prototyping card with wire-wrapped connections. The card contains the interface to the IBM addresses and data buses, circuitry to control the signal-conditioning unit, and the A/D sequencer.

IBM Bus Interface

Figure 10 shows the block diagram of the calibration IBM interface card. (The schematic is shown in figures 29-32 in appendix A. Figure 29 shows all of the bus interface.) Two one-of-eight decoder/demultiplexer chips (74LS138) provide the address selecting for the card (address range #300 to #31F). An octal bus transceiver (74LS245) buffers the data lines and a octal buffer/line driver (74LS244) supplies the bus control signals. Finally a dual D-type flip-flop (74LS74) connects the output of the A/D sample counter to the PC interrupt controller.

Signal-Conditioning Unit Interface

The circuitry diagrammed on the right side of figure 10 constitutes the signal-conditioning unit interface. (Schematics are in appendix A.) All digital interface lines are opto-isolated to allow flexible power-up sequencing and to prevent ground loops in the analog interface. An 8255A Programmable Peripheral Interface chip provides both the address lines and the control signals for the signal-conditioning unit. The control signals (D/A enable, D/A strobe, cal select, and shunt cal select) are taken from port C of the 8255A because these lines are individually addressable. One of the counters in the AM9513 chip generates the clock signal for the anti-aliasing filters on the analog signal-conditioning cards. Finally an eight-bit latch (74LS273) interfaces the PC data bus to the signal-conditioning unit data bus.

A/D Sequencer

The A/D sequencer is the most complex part of the calibration IBM interface card, and the AM9513 system timing controller is the major component on that card. This chip includes five general-purpose, 16-bit counters. A variety of internal frequency sources and external pins may be selected as inputs for individual counters with software selectable

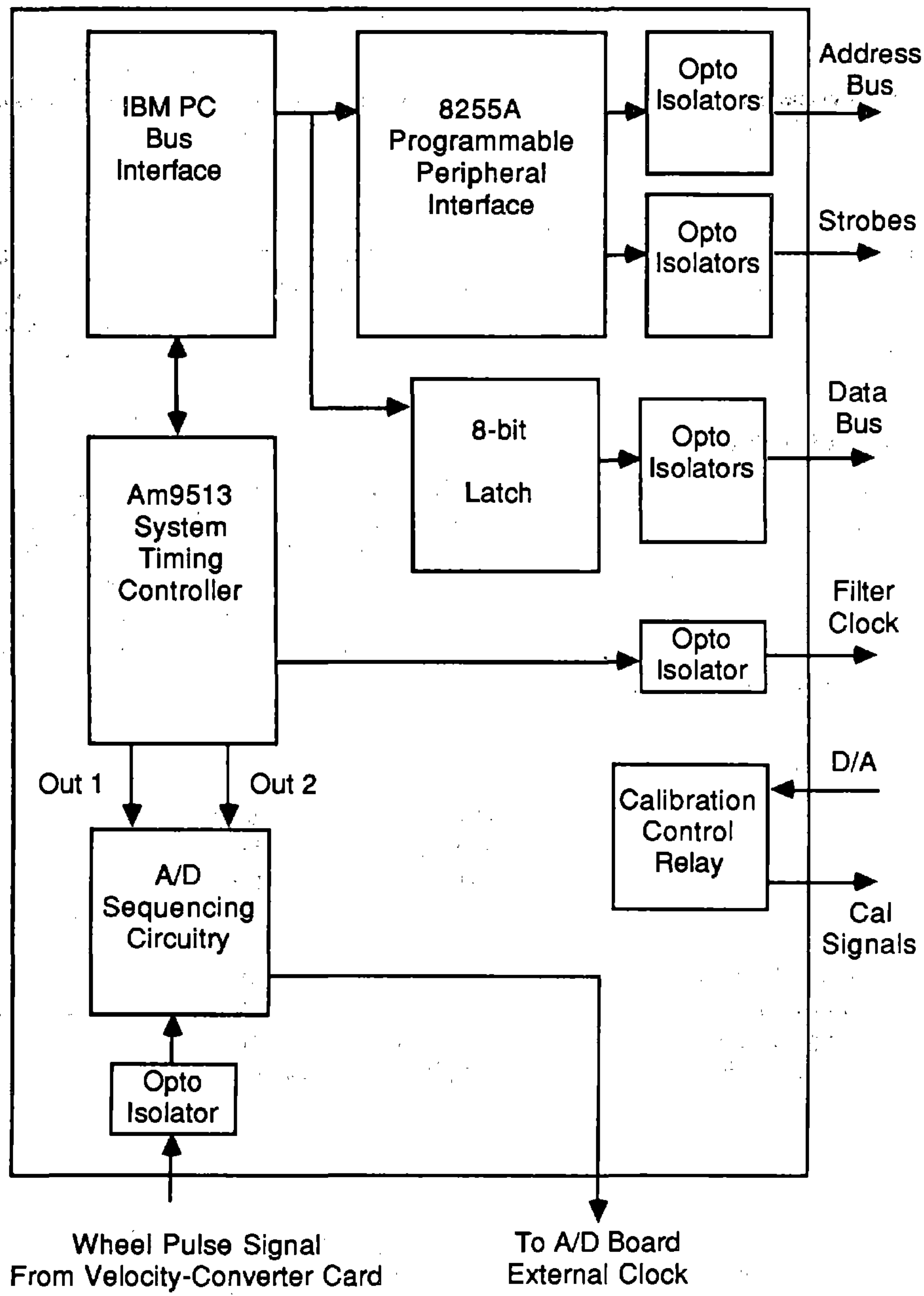


Figure 10. Block diagram of the calibration IBM interface board.

active-high or active-low output polarity. Both hardware and software gating of each counter is available. The counters can be programmed to count up or down in either binary or BCD. The output from any of the counters can be connected via software to the input of another.

Table 1 and figure 11 illustrate how the counters are used in the profilometer. The table indicates the mode, direction, source, output type, and initial counter value for each of the five counters. Figure 11 translates the above into the mode register bit assignments which actually program the counters. Counter #1 is used to count down either a time-based clock signal derived from the system clock or the signal from the wheel pulser via the velocity-converter card. The output from this counter synchronizes the beginning of an A/D scan with elapsed time intervals in the bounce test or distance intervals along the road in the profile test. Counter #2 provides the control of the individual channel sampling (set to the maximum rate for the Data Translation A/D board) within the A/D scan. Counter #3 counts down the output of counter #2 to provide a gating signal. For example, if there are three data channels, only three pulses from clock #2 are gated to the A/D. Counter #4 counts each of the samples and gives an output pulse when a buffer is full. This pulse generates an interrupt which causes the DMA controller chip to be programmed with the address of the next buffer. Counter #5 generates the filter clock and therefore is not used in the A/D sequencer.

The A/D sequencer schematic is shown in figure 30 in appendix A, as part of the calibration IBM interface card. Figure 12 shows the timing diagram for the A/D sequencer. The output of counter #2 (OUT2) is used as the clock input for flip-flops U35A and U35B. The output of flip-flop U35A (TRIG1) is the output of counter #1 synchronized to the falling edge of the signal from counter #2. The output of flip-flop U35A is the A/D gate signal. This signal is high to enable the 25.3 Khz. clock into the A/D external clock input. It goes high on the falling edge of the next clock pulse after TRIG1 and goes low when counter #3 counts down (i.e., when all channels in the scan have been sampled). The bottom line of figure 12 shows the output of the A/D sequencer for the case of four channels in an A/D scan.

Exercising the Hardware Using Software

The analog signal-conditioning unit can be exercised using several built-in functions. Unlike other functions offered by the system, these do not have a well defined role in the context of routine testing, checking, or maintenance. They were used during the development of the system, and have been retained as tools that a technician may choose to use as he or she sees fit. They are accessed by choosing the option to EXERCISE INPUT/OUTPUT SYSTEM from the main menu. A new menu then appears on the screen offering the following options:

- *SET CALIBRATION D/A* allows the operator to set the value (±5 volt range) of the calibration signal going into the calibration relays on the analog cards.

Table 1. Counter usage summary.

Counter	Usage	Mode	Direction	Source	Output	Counter Value
1	Distance-based division	D	Down	SRC 2	Positive	IDIV
1	Time-based division	D	Down	F1	Positive	IDIV
2	A/D clock-25.3868 kHz	D	Down	F1	Toggle	47
3	Channel counter	R	Down Neg. edge	SRC 3	Negative	NCHAN
4	Sample counter	D	Down Neg. edge	SRC 4	Positive	NCHAN x NSAMP
5	Filter clock generator	D	Down	F1	Toggle	IFREQ

IDIV=inches/sample / inches/pulse for distance-based sampling
IDIV= 2.386364 µsec / sampling frequency
IFREQ= 1/3 of nominal sampling frequency

Distance-based clock counter

Counter #1	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0221	0	0	0	0	0	0	1	0	0	0	1	0	0	0	0	1

Time-based clock counter

Counter #1	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0B21	0	0	0	0	1	0	1	1	0	0	1	0	0	0	0	1

A/D clock generator

Counter #2	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0B22	0	0	0	0	1	0	1	1	0	0	1	0	0	0	1	0

Channel counter

Counter #3	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
D3A5	1	1	0	1	0	0	1	1	1	0	1	0	0	1	0	1

Sample counter

Counter #4	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
1421	0	0	0	0	0	1	0	0	0	0	1	0	0	0	0	1

Filter clock generator

Counter #5	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0B22	0	0	0	0	1	0	1	1	0	0	1	0	0	0	0	1

Figure 11. Counter mode register bit assignments.

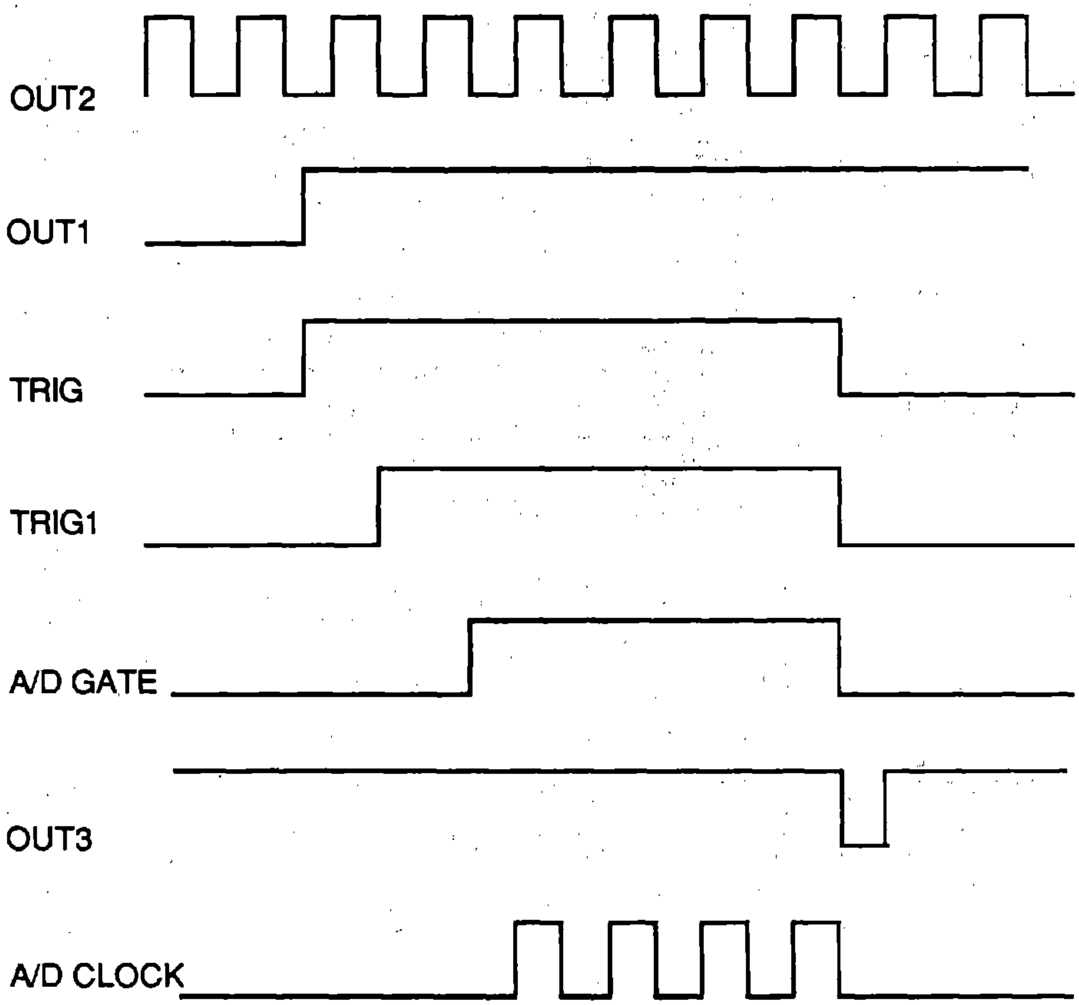


Figure 12. Timing diagram for the A/D sequencer.

- *CALIBRATION RELAY* switches the calibration relay to connect this signal to the amplifier.
- *SET OFFSET* puts in an eight-bit value (±127) into the offset D/A on a card.
- *READ A/D* samples a channel at an operator selected gain and frequency, and prints out the average voltage over the given sampling time.
- *WAIT FOR A SPECIFIED TIME* checks the calendar clock.
- *CLEAR DATA TRANSLATION BOARD* initializes the Data Translation A/D board.
- *SET DATA TRANSLATION CLOCK* sets clock on Data Translation A/D board to a specified frequency.
- *SET FILTER CLOCK* sets the filter clock generator on the calibration IBM interface card to the frequency that the operator inputs.
- *RESTORE ANALOG* turns off all calibration relays and load all of the offset D/A's with their last entered values. This command is used to restore the state of the signal-conditioning unit when power is turned off to enable board removal and insertion.
- *A/D REFERENCE* switches on the A/D reference signal on the A/D check card so that it can be checked with a voltmeter.

One use of these functions is to measure an amplifier gain. To measure the gain of an amplifier, switch the cal relay, put in two known voltages, read the outputs, and calculate the gain based on the intervals.

The bubble disk contains a program written in Basic that can be useful for diagnosing problems with the calibration IBM interface card. Called CNTTST.BAS, it sets up the 9513 chip and the related circuitry for trouble-shooting using conventional laboratory equipment. The listing is included in appendix D.

DATA COLLECTION

Three different methods of data collection are used in the profilometer software. They vary from the simplest, used in the calibration routines, to the very complex, used in the test routine. The following subsections describe these methods and their performance limitations. Considerations involving the sequence used to sample the transducers are also mentioned.

Sampling a Single Channel Using Direct Memory Access (DMA)

Under the first method, the system collects data from one analog channel and requires only the Data Translation (DT) A/D board. The data acquisition parameters of channel number and gain are set via a call to the subroutine SETAD. Next the DT clock is set to the desired sampling frequency through the subroutine DTCLOCK. Then the DMA controller and the DMA page register on the PC motherboard must be set to their appropriate values. Finally the collection begins with the start A/D command. As can be seen in part *a* of figure 13, the samples are taken coincident with the DT clock signal.

A very important limitation of the hardware is that a page boundary cannot be crossed when collecting data via DMA. For this reason, at most 32767 samples can be collected and the buffer address must be carefully chosen.

The subroutine A2DONE, called by the calibration and transducer check routines, uses the above method to collect data from one analog channel. To ensure that a page boundary is not crossed, it finds the physical address (integer*4 ADDR) of the buffer IBUF and then calculates the starting index into the buffer, i.e., IBUF (INDEX), so that the upper word of ADDR and the upper word of the address of IBUF (INDEX+ number of conversions) are the same. The address of IBUF (INDEX) is then loaded into the DMA registers by a call to SETDMA.

Interrupt-Driven Sampling of Multiple Channels

The second method allows collection of up to eight differential channels for as long as there is memory available. (Eight is the maximum number of channels on the A/D board.) The start channel, stop channel, and gain are set via a call to SETAD. Part *b* of figure 13 depicts this operation. The DT clock is set to 20 Khz and the counters in the 9513 chip are used to generate an interrupt at the specified frequency. This interrupt can be based on the system clock or from the wheel pulser. When the interrupt occurs, the channels are sampled in the order as specified above. If there will be a page crossing during a scan, the interrupt routine uses programmed I/O to individually collect each data channel (20 Khz is near the maximum frequency for programmed I/O). Otherwise, it programs the DT board and the DMA controller to collect the channels and store the data via DMA. Because the interrupt latency period is variable, this method does not give as precise a sampling interval

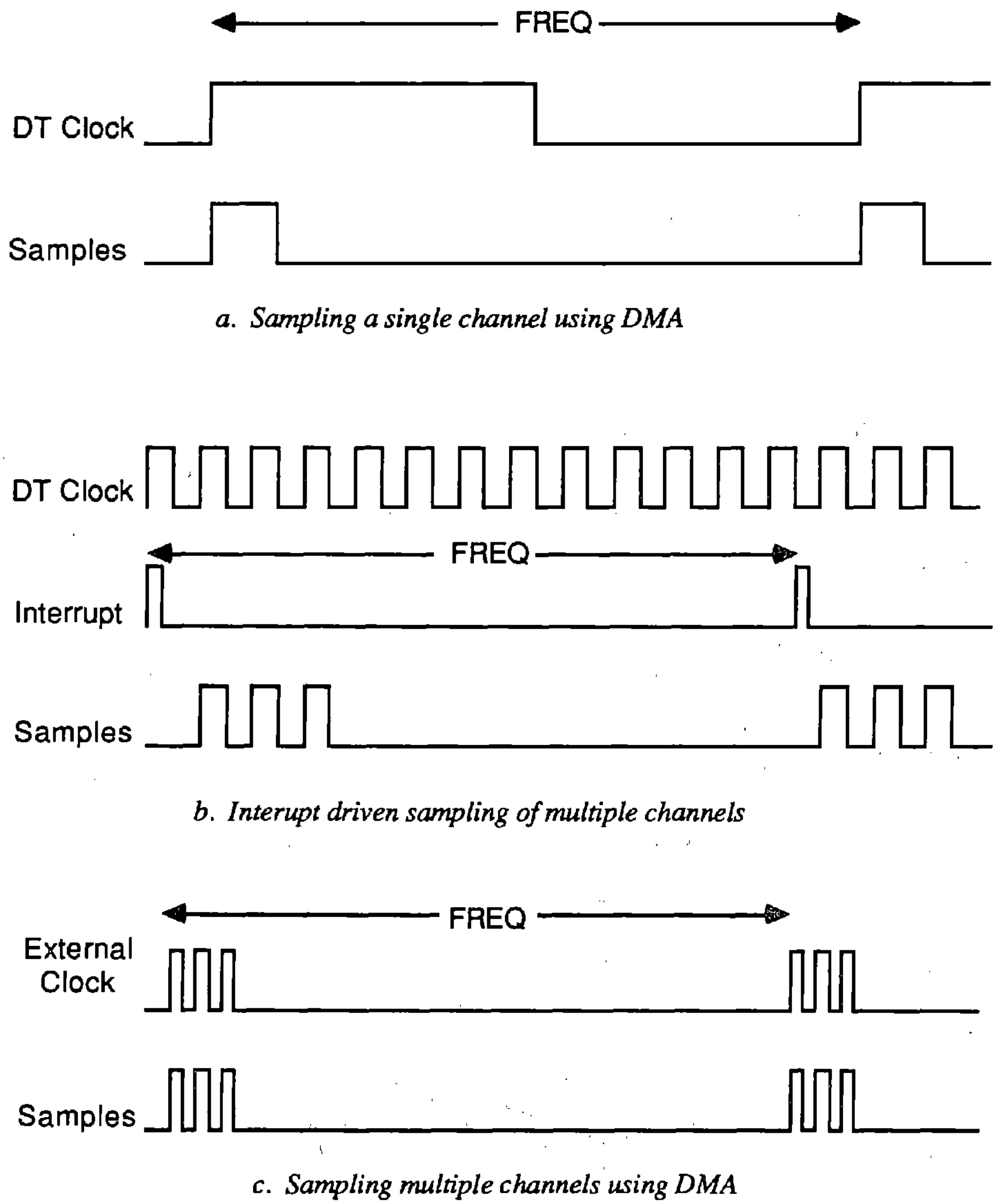


Figure 13. Timing diagrams for three methods of controlling sample rate.

as the DMA method described above. However, it does allow more than 32767 samples to be taken . Also, since much of the CPU's time is consumed by acquisition of the data, simultaneous checking of the data or writing the data to tape is not feasible. This method is used only in the pulser test where the data are not saved to tape.

Sampling Multiple Channels Using DMA

The third method of sampling allows collection of up to eight differential channels (the maximum number of channels on the A/D board) for as long as there is tape storage available. The start channel, stop channel, and gain are set via a call to SETAD as in method #2. In method #3 the DT clock is not used. An external clock generated by the A/D sequencer circuitry synchronizes the acquisition of the data to the system clock (in a bounce test) or to the wheel pulser (in a regular test). Part *c* of figure 13 shows the external clock generated for sampling three channels of data. This could also be accomplished by using the DT clock and an external trigger. However, the maximum sampling rate would be lower because of the time it takes the DT microprocessor to start the A/D after an external trigger. Thus the DMA method is the most efficient way to collect multiple channels for long periods of time.

Figure 14 shows how buffers in memory are used for collecting data using this method. The buffer array is divided into 15 buffers of 16384 bytes each. The buffers always include an integer number of complete scans. For example, with three channels of data, only 16380 bytes of the 16384 bytes are actually used. As with the single-channel DMA method, each of these buffers must not cross a page boundary.

The A/D starts by filling buffer #1. When the buffer is full, an interrupt is generated and the DMA controller is set to point to the next buffer. At this time, the status of the next buffer is checked. If it is already full, data collection is necessarily terminated to prevent a loss of previously acquired data. Otherwise, the A/D continues to fill the buffers , one by one, progressing from buffer #1 to Buffer #15 and then back to buffer #1. While the buffers are being filled via DMA, the test software monitors their status. When a buffer is full, it is written to tape and its status is cleared. Normally, the buffers are written to tape and cleared shortly after they are filled, such that only a few buffers are in use at any given time. Under these conditions data collection can continue until the segment of tape is full.

Sometimes the tape software must update the directory on the tape, causing the tape writing to fall behind the data collection. This increases the likelihood that a full buffer will be encountered and that the test must be terminated prematurely. As might be expected, the system has an easier time keeping up when the incoming rate is low, as occurs when only a few channels are sampled or when they are sampled at a low rate as occurs at low test speeds. (Tests of over 10 miles have been done sampling five channels every three inches.)

Due to a quirk in the design of the Data Translation board, the process of collecting data is actually more complicated than indicated in the above discussion. The Data Translation

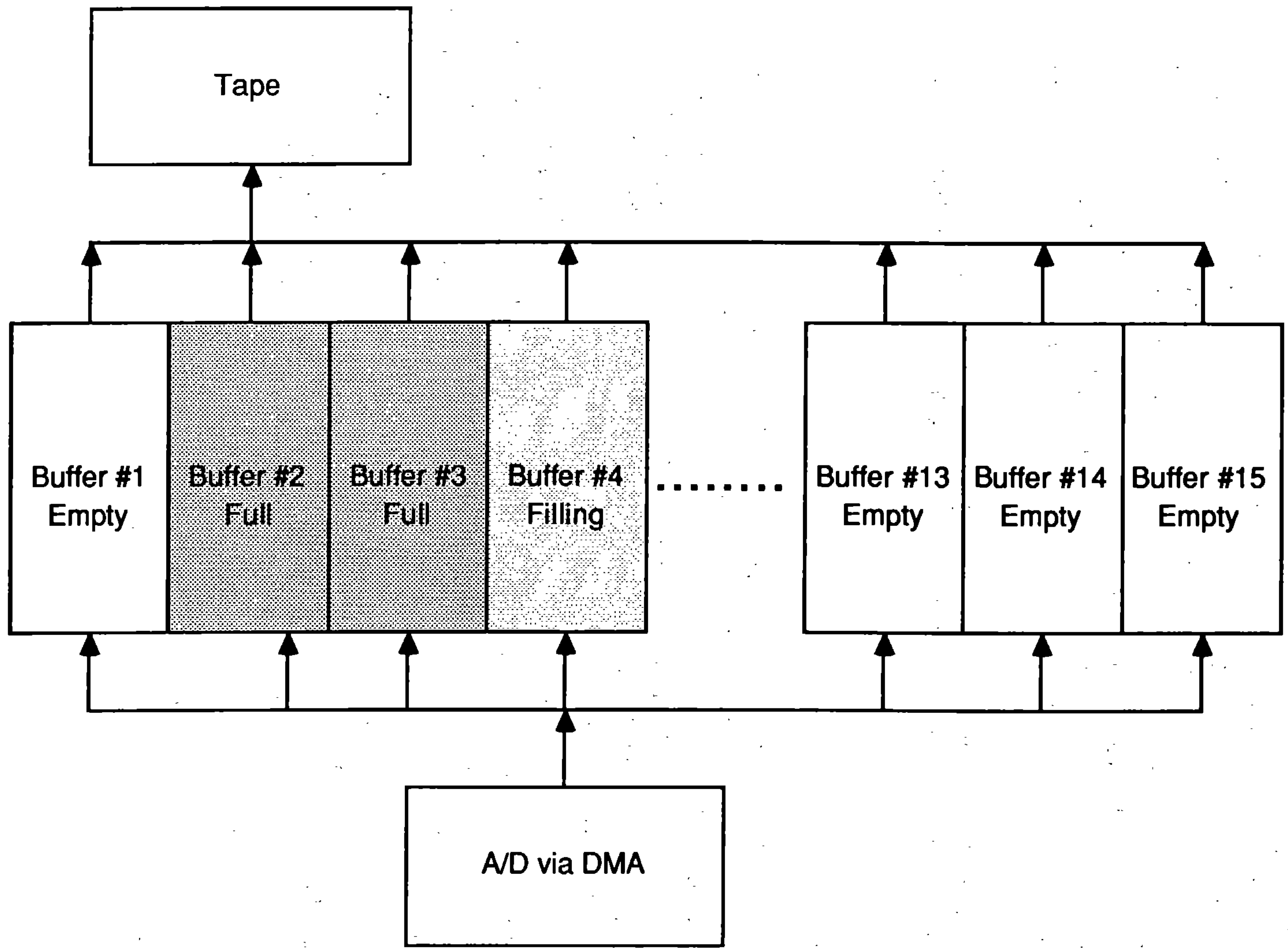


Figure 14. Memory buffers used for collecting data.

board samples coincidently with the external clock, but it does not store the last byte of the sample until the next clock pulse. This causes the last byte of a buffer to be stored at the beginning of the next buffer. To consider the effect this has on the data collection software, consider the the first three buffers in a three-channel test, shown in figure 15. The array BUFT contains the addresses that are loaded into the DMA controller by the interrupt software. The array BSTRT contains the buffer beginnings that are used by the write-to-tape routines. As can be seen in the figure, the high byte of the last sample of buffer #1 is actually at the beginning of buffer #2. Since the buffer array is of integer*2 type, only a word address can be passed to the write routine. Thus, buffer #1 is not written to tape until buffer #2 is full. The remaining byte (the high byte of the last sample of the last channel) is moved into position in buffer #1, and then buffer #1 is written to tape.

After buffer #1 is written to tape the first time, the pointer in BSTRT(1) is incremented because in the next pass the high byte of the last sample of buffer #15 will occupy the first location of the buffer.

Valid Configurations

The Data Translation board allows eight differential channels to be sampled. The channels that are sampled are determined by a starting and stopping index, such that all channels between the start and stop are sampled. For example, it is possible to sample channels 0, 1, and 2 by specifying the range 0 to 2; it is also possible to sample channels 5, 6, 7, and 0 by specifying the range 5 to 0. However, the hardware does not allow the sampling of channels 1, 4, and 5 because they are not contiguous.

Not all combinations of channels can be used to advantage with a road profilometer. For example, there are no measures that can be obtained from two height signals that do not include the vehicle response. Given the objectives of measuring longitudinal profile and rut depth, the eight transducers for the profilometer have been assigned the permanent channels indicated in figure 16. This layout puts the three sensors (height, acceleration, and velocity) needed for longitudinal profile adjacent to each other in positions 0 to 5, so that either profile can be measured efficiently by sampling only three channels. Figure 20 (contained in the section "Data Files") shows the valid configurations allowed for the system along the channels that are sampled by the digitizer for each configuration.

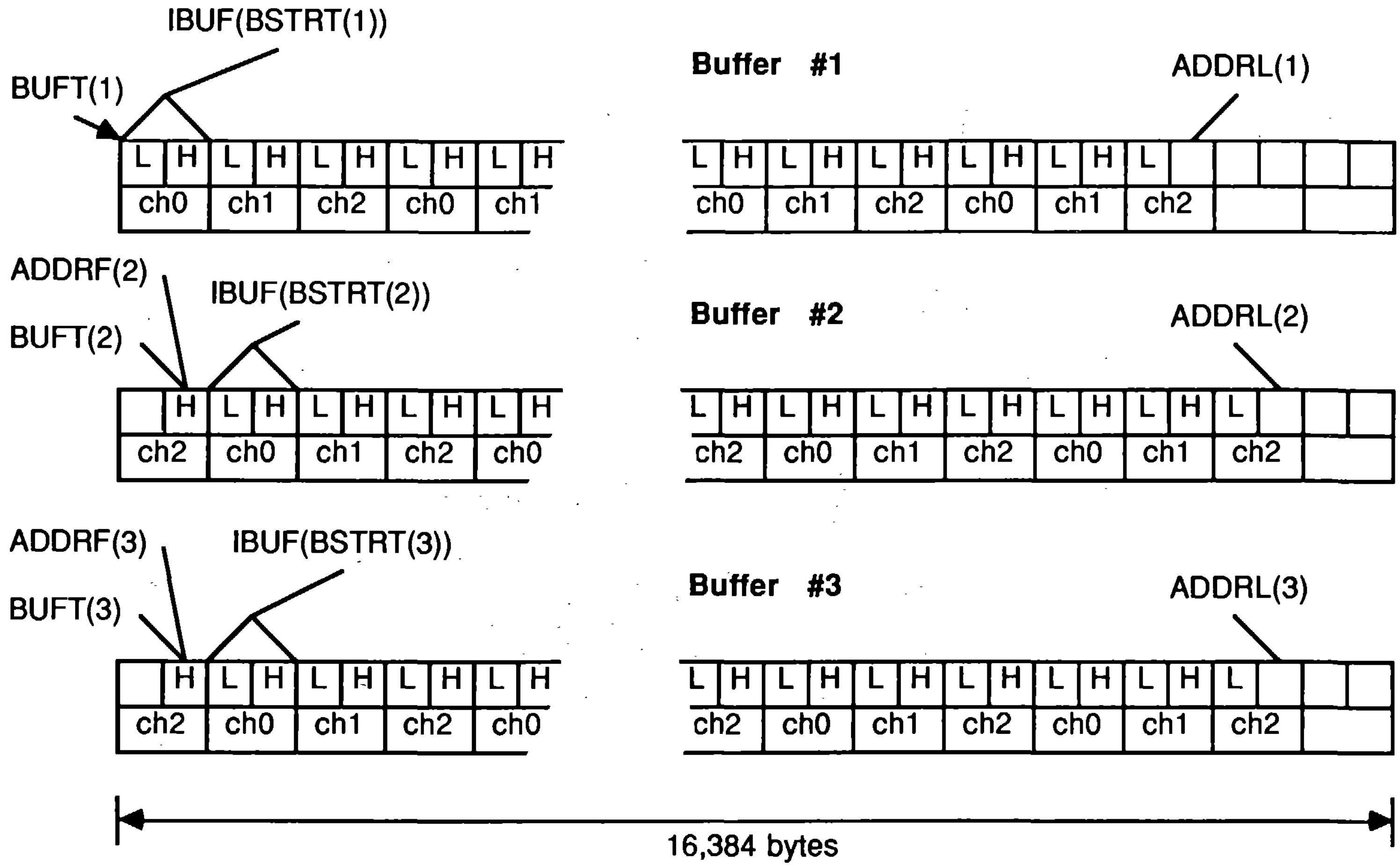


Figure 15. Illustration of carryover byte between buffers when collecting data

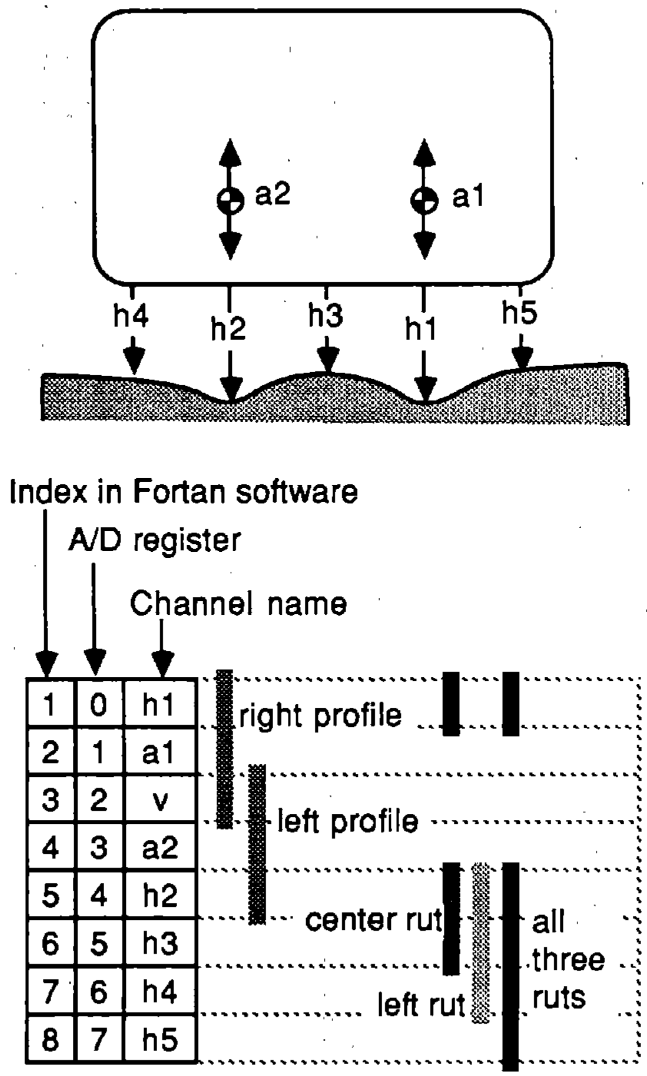


Figure 16. Schematic layout of transducers in the profilometer.

COMPUTATION METHODS

This section describes the mathematical transforms used to convert signals from the height sensors, accelerometers, and speed sensor into slope profiles, elevation profiles, roughness levels, and profiles of rut depth. Because all of the equations are applied in the Fortran language, the equations are shown using Fortran notation. The many Fortran functions and subroutines that make use of these analyses are described in the later section, *Profilometer Subroutines*.

In addition to the theoretical considerations of computing the desired measures from the transducer signals, there are also practical issues to face when performing the calculations on a computer with memory limitations. This section describes how buffers are used to allow the data files to be much larger than the computer memory.

Equations and Signal Processing

Slope Profile

The UMTRI/FHWA profilometer computes longitudinal profile using a variation of the method invented by Spangler and Kelley at the General Motors Research Laboratory.[7] Three measured signals—acceleration, height, and speed—are combined to yield the profile of the road. For several technical reasons, the slope profiles are stored on tape rather than the elevation profiles. The computation of slope profile includes five steps:

1. the bias in the accelerometer signal is calculated and subtracted to minimize error in the following integration;
2. the acceleration signal is converted from temporal acceleration to spatial acceleration;
3. the spatial acceleration is integrated once to obtain a slope signal;
4. the height signal is differentiated once to obtain a slope signal; and
5. the slope signals from the height and accelerometer sensors are added to obtain the slope of the profile.

These steps are accomplished numerically for signals that have been digitized at a constant spatial interval. The equations are shown below, using Fortran notation similar to the computer code used in the subroutine PRFCMP described in the section *Profilometer Subroutines*.

The first step is straightforward, and is accomplished using a Fortran function called RAVE. The second step is achieved with the equation:

```
ACCS (I) = SCALE * ACCT  (I) / SPEED (I) ** 2                    (1)
```

where

I	=	sample number (1, 2, ...)
ACCT (I)	=	i-th sample of accelerometer signal (temporal, with units such as m/sec^2)
ACCS (I)	=	i-th sample of spatial acceleration (with units such as 1/m)
SPEED (I)	=	i-th sample of vehicle speed (with units such as m/sec)
SCALE	=	scale factor needed to obtain correct units in eq. 1. (If the signals have the m-sec units indicated in parentheses, then SCALE = 1. In the present software, SCALE is calculated from scale factors relating the units used for the transducers to m-s equivalents.)

The third step is achieved with a digital filter, defined by the recursive equation:

S1 (I) = COFINT * S1 (I - 1) + ACCS (I) * DELTAX (2)

where

S1 (I)	=	component of slope profile obtained from the accelerometer (m/m)
COFINT	=	constant filter coefficient slightly less than 1.000, defined below in eq. 5 (dimensionless)
DELTAX	=	sample interval (with units of m)

The fourth step is achieved with a digital filter that has identical phase properties as eq. 2, but serves to differentiate rather than to integrate:

S2 (I) = (COFINT * H (I + 1) - H (I)) / DELTAX (3)

where

S2 (I)	=	component of slope profile obtained from the height sensor (m/m)
H (I)	=	i-th sample from the height sensor (with units of m)

The complete profile is the sum of the two components:

SP (I) = S1 (I) + S2 (I) (4)

where

SP (I)	=	slope profile (m/m)

The coefficient COFINT should be given a value slightly less than 1.0000. A value of 1.0000 means that the integrator and differentiator defined by eqs. 2 and 3 do not include any additional filtering to remove d.c. drift and very long wavelengths. In the PRFCMP subroutine, the value of COFINT is determined by the equation:

COFINT = 1 - DELTAX / LNGWAV (5)

where

LNGWAV = A spatial equivalent of a time constant, which will be about 1/3d of the longest wavelength of interest (m)

Rut Depth

Rut depth is computed from three height signals, as shown in figure 17. The rut depth of a wheeltrack is the difference between the elevation in the wheeltrack compared to a line drawn between two reference points on either side of the wheeltrack. Alternatively, a *middle rut* profile is available that shows the average difference in elevation of the two wheeltracks and a single reference point located between them. For each point, the rut depth is computed using the relation:

$$R(I) = [(LL \times HL(I)) + (LR \times HR(I))] / (LL + LR) - HC(I) \quad (6)$$

where

R(I) = i-th value of the computed rut depth
LL = distance from left-hand height sensor to center height sensor
LR = distance from right-hand height sensor to center height sensor
HL(I) = i-th sample from the left-hand height sensor
HR(I) = i-th sample from the right-hand height sensor
HL(I) = i-th sample from the center height sensor

These calculations are performed by the subroutine RUTCMP. The subroutine calculates the rut depth for every sample and accumulates those results over ten samples. The average is calculated and kept for later writing to to tape file.

Roughness

Roughness is computed using a quarter-car simulation using standard vehicle parameters and a standard simulated speed of 50 mi/h. The measure is called the *International Roughness Index* (IRI). The quarter-car simulation involves four variables that define the computed motions of a reference vehicle. At each point along the profile, each of these four variables are calculated using the equations:

$$\begin{aligned}
X1(I) &= X1(I\text{-}1) * S11 + X2(I\text{-}1) * S12 + X3(I\text{-}1) * S13 + X4(I\text{-}1) * S14 + P1 * SP(I) \\
X2(I) &= X1(I\text{-}1) * S21 + X2(I\text{-}1) * S22 + X3(I\text{-}1) * S23 + X4(I\text{-}1) * S24 + P2 * SP(I) \\
X3(I) &= X1(I\text{-}1) * S31 + X2(I\text{-}1) * S32 + X3(I\text{-}1) * S33 + X4(I\text{-}1) * S34 + P3 * SP(I) \\
X4(I) &= X1(I\text{-}1) * S41 + X2(I\text{-}1) * S42 + X3(I\text{-}1) * S43 + X4(I\text{-}1) * S44 + P4 * SP(I) \quad (7)
\end{aligned}$$

where

X1(I)...X4(I) = 4 vehicle variables at the i-th position on road
S11...S44 = 16 coefficients that are called a *state transition matrix*

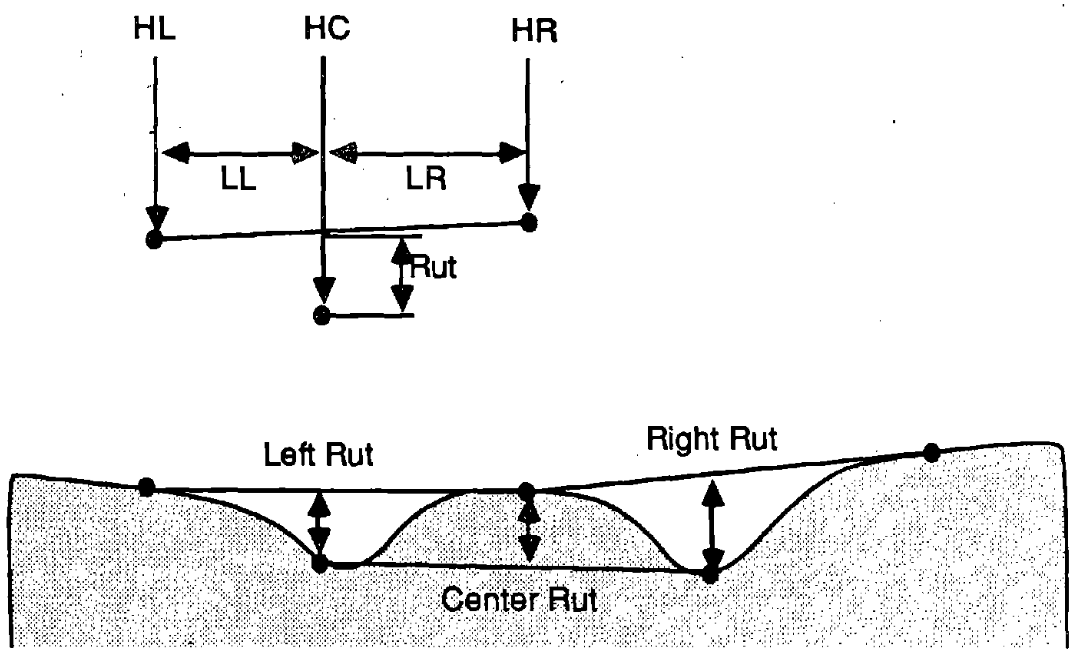


Figure 17. Definition of rut depth used in the profilometer.

P1...P4 = 4 coefficients that are called a *particular response matrix*
SP(I) = i-th value of slope profile

The 20 coefficients used in eq. 7 are a function of the sample interval, DELTAX. They are calculated when the operator chooses the sample interval, using the subroutine SETSTM. The method used in SETSTM for computing these coefficients is described elsewhere.[8]

The roughness is accumulated using the Fortran line of code:

ROUGH (I) = ROUGH (I - 1) + DELTAX * ABS (X1(I) - X3(I)) (8)

The roughness is updated at every sample of slope profile, but only every tenth value is stored on the tape. These calculations are performed in the Fortran subroutine PRFIRI.

Profile Elevation

The profile elevation is computed from the profile slope using the same digital filter uses to integrate the accelerometer in eq. 3. The integration of slope is performed backwards to cancel the phase lag introduced when computing slope via eqs. 1 - 4. When moving backwards (from the end of the test to the beginning) the numerical integration is defined by the equation:

EP (I) = COFINT * EP (I + 1) + DELTAX * SP (I) (9)

where

EP (I) = i-th value of the elevation profile.

This profile signal has no phase distortion introduced by the data processing. This means that the same profile should be measured regardless of the direction that the profilometer is travelling over the wheeltrack. The elevation profile stored in the data file is computed by the PRFCMP subroutine. The GETELV subroutine—used to get elevation data for plotting—also computes elevation using eq. 9 when detailed plots are requested by the user.

Filtering with a Moving Average

The elevation profiles are always filtered to remove long wavelengths when they are plotted. The filtering is accomplished by the subroutine HIPASS using a moving average. A moving average is also used by the subroutine LOPASS to smooth the roughness and rut depth profiles.

The moving average involves averaging an input signal over a number of samples to obtain each value of the output signal, using the equation:

$$y_s(i) = \frac{1}{m} \sum_{j=i-k}^{i+m-k} y_r(j) \qquad (10)$$

$y_r(j)$ = j-th value of original (raw) signal
$y_s(i)$ = i-th value of smoothed signal
m = number of samples in the moving average baselength
k = number of samples in 1/2 of the moving average baselength
b = $m \times \Delta$ = baselength of moving average
Δ = distance between samples

In order for eq. 10 to duplicate a true moving average (as occurs in the limit when Δ approaches zero), the value of m should not be too small. A value of m=9 points in the summation is a reasonable lower limit. As m increases, such that the baselength is much longer than the sample interval, the equation approaches a true moving average.

The computations implied by eq. 10 are written more efficiently for the computer software:

$$y_s(i) = y_s(i-1) + \frac{1}{m} [y_r(i+m-k) - y_r(i-k-1)] \qquad (11)$$

Eq. 11 is recursive, meaning that the new value for $y_s(i)$ depends on the previous value, $y_s(i-1)$. This equation is much more efficient than eq. 10: even if the moving average includes thousands of points, each smoothed value is calculated from just two of the original values (at sample numbers i+m-k and i-k-1) and the previous smoothed value.

The moving average is converted from a lopass filter to a hipass filter by subtracting the smoothed signal from the original signal:

$$y_h(i) = y_r(i) - y_s(i) \qquad (12)$$

where y_h is the hipass filtered signal.

Profile elevation is filtered using eqs. 10 through 12 to remove long wavelengths whenever profile plots are made, using the Fortran subroutine HIPASS. Using only these equations, the first k and last m-k values cannot be plotted. This is because eq. 11 requires an initialization to obtain the first value of the smoothed signal, and it also "looks ahead." In order to show the entire filtered profile, including the first and last k points, artificial data are added automatically by the software at the beginning and end of the measurement. The extra points are generated using the equation:

$$y_a(i) = y_r(1) + \bar{y}' \cdot (i-1) \qquad (13)$$

where

$\bar{y}'$ = slope of profile (with respect to sample number) for the first k samples
$y_a(i)$ = artificial profile point
i = 1-k ... 0 ($i \le 0$)

Eq. 13 generates additional points that lie on a straight line which connects to the elevation of the first point of the measured profile. The slope $\bar{y}'$ is computed by a linear regression between elevation and sample number over the first k samples.

The same method is used to generate artificial points at the end, using the equation

$$y_a(i) = y_r(n) + \bar{y}' \bullet (i - n) \tag{14}$$

where

$\bar{y}'$ = slope of profile (with respect to sample number) for the last m-k samples
i = n+1 ... n+m-k

These artificial points are created as needed, based on the interval of profile to be plotted and the current baselength for the moving average. They are never stored in the data file.

Plotting of Elevation

Two methods are available for plotting elevation profile. These are available to the user as *quick* and *detailed*. When the *quick* option is selected, the data are read from the elevation part of the data file. The profiles have units of height, and are filtered with the hipass moving average and plotted. When the *detailed* plotting is requested, the slope profile data are read into memory and integrated backwards using eq. 9 to obtain the detailed elevation profile. The first elevation value, used to initialize eq. 9, is obtained from the elevation part of the data file. Thus the slope profile is always read up until the next distance for which an elevation point is stored in the file. The elevation data is used to ensure that all elevation profiles that are computed have the same reference, which is an (arbitrary) elevation of 0 at the end of the run. By using the precomputed elevation data, the slope profile can be integrated starting at any point in the file with the same result as would be obtained by starting at the end of the file and integrating all the way back to the data of interest. The Fortran subroutine GETELV is used to transfer the data from the file to memory and to perform any necessary processing to obtain filtered elevation profiles. If a detailed plot is requested, the subroutine loads the detailed slope profile and performs the backwards integration. For either type of profile, the HIPASS subroutine is called to add any necessary artificial points and apply the hipass moving average filter.

Buffers and Memory Usage

Rut Depth and Roughness

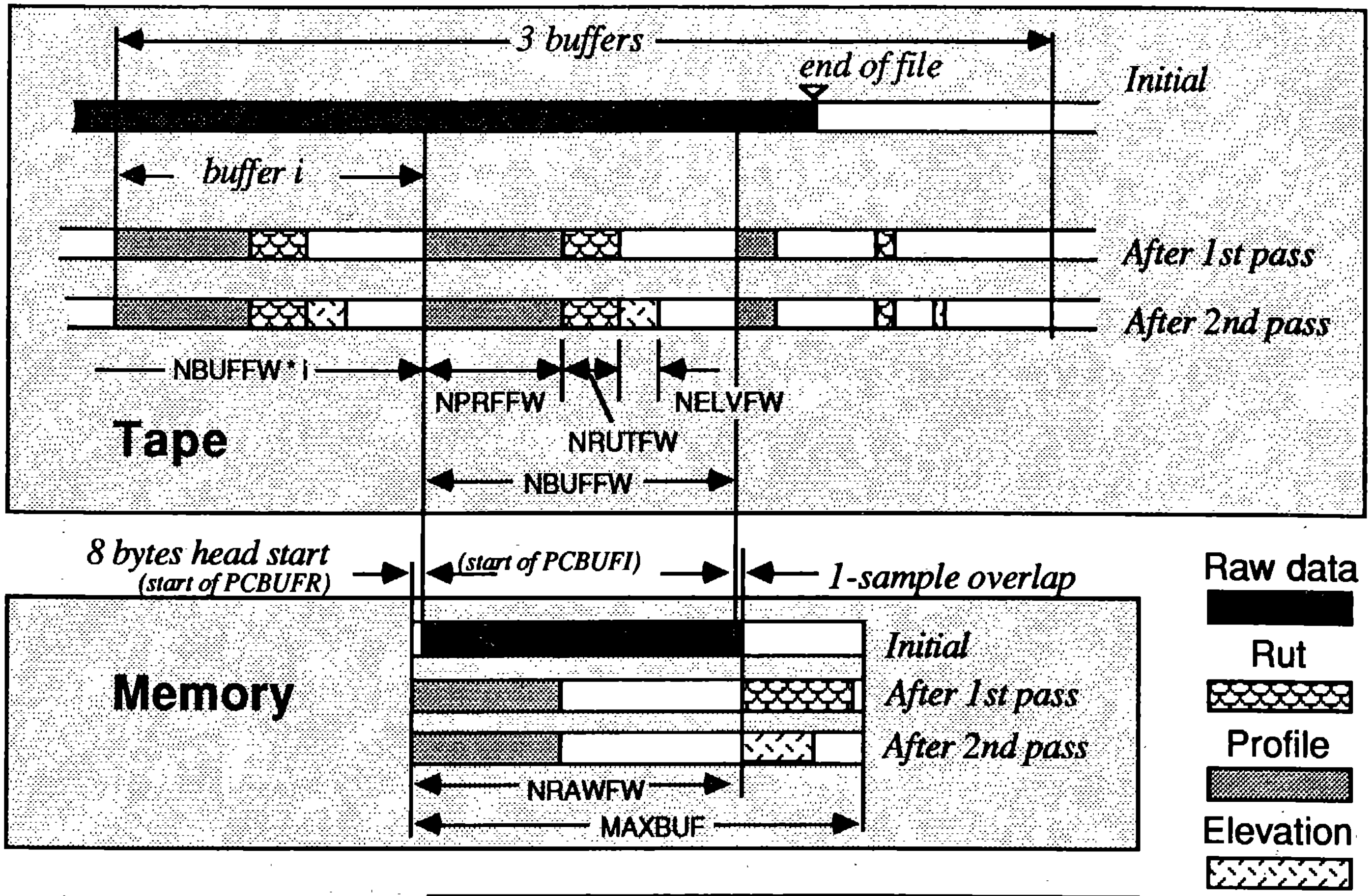
The profilometer software reserves some of the memory of the machine for the storage and processing of the signals measured with the profilometer. The available memory, shown in figure 18, is specified by the Fortran parameter MAXBUF. This memory is divided into two sections—one to hold the raw data, and one to hold the rut depth and roughness data. Both the rut depth and roughness signals are decimated by a factor of ten, and therefore less memory is needed for the computed signals. (The decimation factor is stored as the Fortran variable TRIM in the file header. The software will also work if a different value for TRIM is used, but figure 20 in the next section, *Data Files*, should be consulted to ensure that there will be room for the data in the file if a TRIM value smaller than ten is used.) The speed signal is averaged and decimated by a factor of ten and the decimated signal is also stored in the region of memory used for the rut depth and roughness signals. The size of the region reserved for the raw data is given by the Fortran variable NRAWFW and the size of the region reserved for the computed data is given by the variable NRUTFW. Figure 18 shows the equations used to calculate the sizes of the two regions as functions of the number of channels in each.

The averaged rut depth and speed signals depend only on the raw data currently in memory, so the computation is straightforward. The roughness is computed by marching through the data, calculating new values for the variables in the quarter-car simulation from the profile and from the previous values of the quarter-car variables. The values of the quarter-car variables are preserved between buffers.

Slope Profile

The PRFCMP subroutine uses the same memory for storing the raw data signals (input) and the computed slope profile (output). Once the slope profile is computed, the raw signals no longer exist. (Thus the rut depth calculations must be made before the slope profile calculations.) Replacing the input data with the output data is a little tricky for several reasons. First, the input data are integer*2 numbers, while the output data are real*4 numbers. Second, the number of channels in the raw data is not the same as the number of profiles being computed. Third, eq. 3 requires two consecutive samples from the height signal. It is important that none of the raw data values get overwritten until after they are no longer needed. To ensure this, the memory areas used for the input and output arrays are not exactly the same.

Figure 18 shows the relative positions of the data arrays in memory, and how those arrays relate to the tape file. After processing, the tape file is divided into segments spaced by NBUFFW *reals* (4 bytes = 1 real*4 number) from the start of one segment to the start of the next. The amount of raw data read into memory for processing is slightly greater to include one extra sample of each channel from the next buffer. This is needed in order to



Sizes are in units of reals (4-bytes)

Relations used to compute buffer sizes

NRAWFW + NRUTFW ≤ MAXBUF

NRAWFW = 2 + [NCHRAW + (NRSAMP • TRIM + 1) + 1] / 2

NRUTFW = NCHRUT • NRSAMP

Figure 18. Buffers used by the subroutine PRFCMP.

apply eq. 3 to compute the final slope profile samples in the buffer. Figure 18 shows the extra data as a "1-sample overlap." The raw data values are stored in the integer*2 array PCBUFI, and the computed profile values are stored in the real*4 array PCBUFR. As shown, PCBUFI begins eight bytes (two real*4 numbers) after the beginning of PCBUFR. Thus the first two elements in PCBUFR can be set without affecting any of the PCBUFI data. (The 8-bytes offset is defined in an *include file* called SETCOM, described in the section *Profilometer Subroutines*.

Profile Elevation

The profile elevation is obtained using a backwards integration and thus the computation cannot begin until the slope profile has been completely finished. Hence, the data processing takes place in two passes: in the first pass the slope profile, rut depth, roughness, and averaged speed signals are computed and written to tape, replacing the raw data. In the second pass, the slope profile is read from tape and integrated backwards to yield the elevation profile, which is then written to tape. As indicated in figure 18, the same memory locations used for storing the rut depth and roughness data are used in the second pass to store the elevation data. The slope profile is put into the same place memory as during the first pass. The elevation data always take less space than the rut and roughness data, so there is no danger of overflow in using the NRAWFW and NRUTFW buffer sizes calculated earlier.

Filtering with a Moving Average for Plotting

The moving average is used for smoothing the rut depth and speed signals. The plotting range available to the user excludes the first k and last m-k points from the file, as required by eqs. 10 and 11. The data are read from the file using the subroutine RDTAPD and placed in the (large) array in common, PCBUFR. The signals are filtered using the LOPASS subroutine, which overwrites the data, replacing the original signals with smoothed signals. If there are NCHRUT channels, the data should begin at element NCHRUT + 1 in the array. The filtered data will be put into the array starting at the first element. The first samples of the original signals are needed to compute the second sample of the filtered signal (eq. 11), but are not needed after that. Thus values of the unfiltered signals are overwritten as soon as they are no longer needed, and the filtered signals will begin at the start of the array where they are accessed by the plotter.

The moving average is also used by the subroutine HIPASS to remove long wavelengths from the elevation signals. The user is allowed to plot all points that were measured, from the first to the last. This requires that up to k artificial points be added to the beginning of the profile when the plots include the start of the data, and that up to m-k points be added to the end when that is plotted.

Figure 19 shows that the available computer memory is divided into five regions. (In the Fortran subroutines, MOVAV1 is the number of points included in the moving

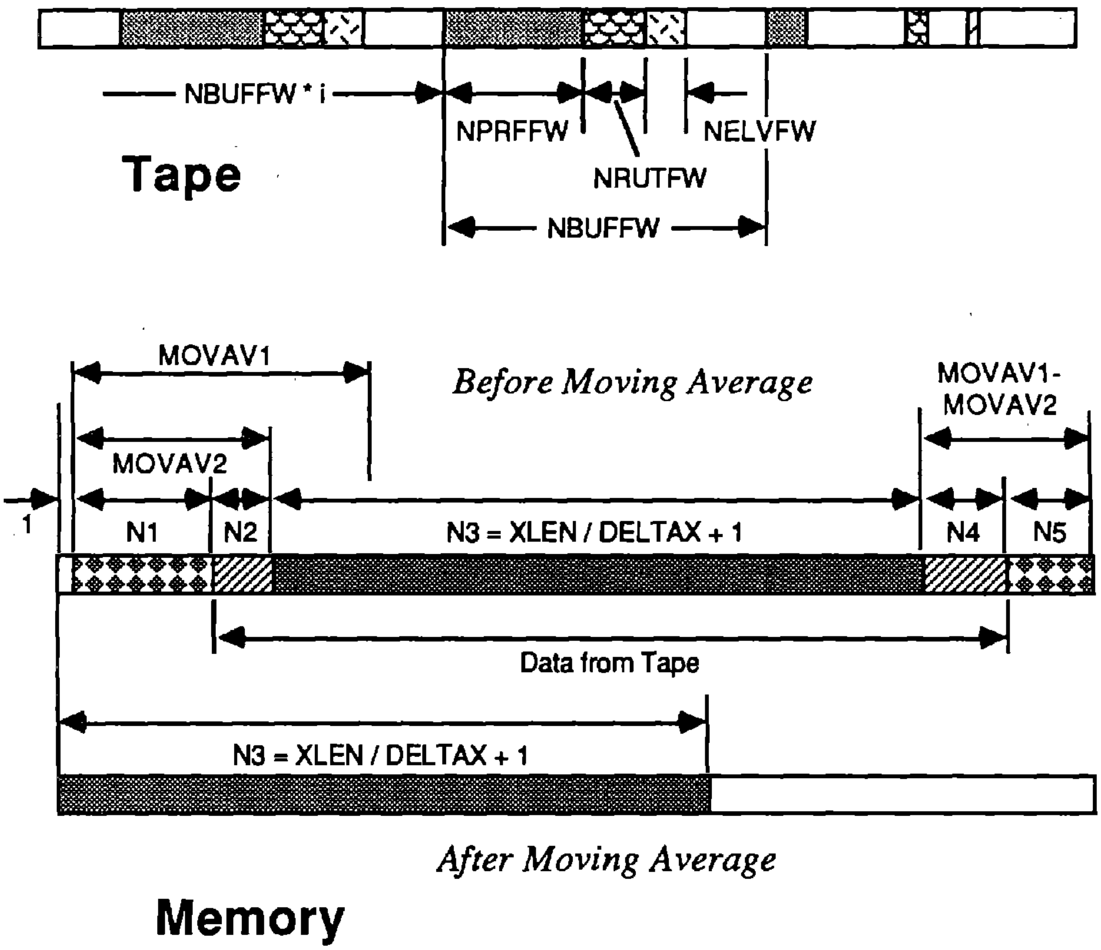


Figure 19. Memory for plotting elevation filtered with moving average.

average—m in eqs. 10 and 11—and MOVAV2 is the number of points to the center of the average—k in the eqs.) N3 is the number of samples needed to show the range requested by the user. N2 and N3 are additional samples of measured profile on either end of the requested range that are needed for the moving average. N2 and N4 can have values between zero and 1/2 the number of samples in the baselength of the moving average. N1 and N5 are additional samples of artificial data, generated by extrapolating the measured profile with a linear regression. They will also have values ranging between zero and 1/2 the number of samples in the baselength of the moving average. The total of N1 and N2 is MOVAV2 (k in the eqs.) and the total of N4 and N5 is MOVAV1-MOVAV2 (m-k in the eqs.). The data points in the regions N2, N3, and N4 are obtained from the data file by the subroutine GETELV. The artificial points in the regions N1 and N5 are added if necessary by the subroutine HIPASS, which also applies the moving average.

As indicated in the figure, the unfiltered data are placed in memory with space at the beginning for one sample of each profile signal. That space is used for the first sample of the filtered signals. The values of the following samples overwrite the unfiltered data as shown, such that when the filtering is complete the signals to be plotted begin at the beginning of the array and contain the correct number of points (N3).

DATA FILES

File Types

A file containing data measured with the profilometer goes through three stages. First, the transducer signals are stored in their original form during measurement. Second, the signals are checked to validate the run. Third, the validated file is processed to compute profiles of slope, elevation, roughness, and rut depth. The original file is modified by the processing, such that the raw data are overwritten when the profile and rut depth signals are computed.

In addition to the normal road test, a special bounce test can be made with the profilometer at rest. The layout and structures of the data files for bounce and road tests are identical.

The files containing road data end with the IBM extension .DTA, and the files containing bounce data end with the extension .BNC. When the user opens a file using the profilometer software, the names of all files having the appropriate extension are shown on the display screen. The status of each file (raw, checked, or processed) is determined after the file is opened.

File Structures

The Header

The first 2048 bytes in the file define a header that contains information describing the test conditions and the layout of the remaining portion of the file. This information is accessed by the profilometer software through an integer array SET contained in the common block SETCOM, described later in the section *Profilometer Subroutines.* A variable named TSTTYP is included in the header and defines the status of the file.

When the file is first created, the header contains the number of data channels, the sample interval, scale factors, names of channels, the date, and many other pieces of information related to the configuration of the profilometer and the type of measurement that is about to be made. Many other variables are set and modified after testing and during processing.

After the header comes the data part of the file, containing the sampled signals measured by the profilometer. During the measurement, the signals from the transducers are digitized and written to tape in the sequence they are taken. When these raw signals are processed, the raw data are overwritten with new signals as described in the section *Computation Methods*. The header is modified to include additional information related to the new layout of the file after it has been converted. The size of the data portion remains fixed, but the structure changes when the raw data are converted to processed data.

The Fortran subroutine UPDSET updates the information in the header of an open file by replacing the SET array as recorded in the file with the current version of the SET array in memory.

Raw Data files

During measurement in both road tests and bounce tests, signals from the transducers are digitized and written to tape in the order that they were taken. Each sample takes 2 bytes and is accessed in Fortran as an integer*2 variable. The first NCHAN[1] $\times$ 2 bytes contain the first sampled values for the NCHAN transducers, the next NCHAN $\times$ 2 bytes contain the second sampled values for transducers, and so forth. When the test is completed, the data portion contains at least NCHAN $\times$ NSAMP $\times$ 2 bytes. For relatively long tests, the data part of the file might include extra room at the end to allow for processing requirements. Only the first NCHAN $\times$ NSAMP $\times$ 2 bytes contain valid data, however.

The sequence in which the transducers are sampled depends on the configuration selected by the operator prior to testing, as described in the subsection *Valid Configurations* in the section *Data Collection.* Ten configurations have been defined in the profilometer software, and the configuration number is stored as the variable TCONFI in the header of the file. Figure 20 shows the order in which the transducer signals are sampled and stored on tape for each of the ten configurations. For example, in configuration number 4, the channels are stored in the sequence: h1, a1, v, a2, h2, h3. (The transducer locations were shown in figure 16 in the section *Data Collection, Methods.*)

The digitized transducer signals are integer*2 variables with values between 0 and 4095, inclusive. They are converted to engineering units with the equation:

$$X_{ij} = (G_j \times D_{ij}) - Z_j \qquad (15)$$

where

X_{ij} = i-th sample of channel j in engineering units
G_j = gain for channel j
D_{ij} = i-th digitized sample for channel j (integer between 0 and 4095, inclusive)
Z_j = zero value for channel j

The gains and offsets are contained in the file header.

The data processing applied to convert the raw data into profile and rut depth signals requires that the output signals have an even multiple of ten samples. The processing also requires that a reduction of at least one sample occur. Therefore, up to ten of the raw data

1 NCHAN is the number of raw data channels; NSAMP is the number of samples.

Configuration	Number	1	2	3	4	5	6	7	8	9	10
	Profiles	L. Profile	R. Profile	2 Profiles	2 Profiles, center rut	L. Profile, L. Rut	2 Profiles, center & left rut	3 Ruts	L. Profile, 3 Ruts	R. Profile, 3 Ruts	2 Profiles, 3 Ruts
raw data file	channels	0 h1 1 a1 2 v	2 v 3 a2 4 h2	0 h1 1 a1 2 v 3 a2 4 h2	0 h1 1 a1 2 v 3 a2 4 h2 5 h3	2 v 3 a2 4 h2 5 h3 6 h4	0 h1 1 a1 2 v 3 a2 4 h2 5 h3 6 h4	4 h2 5 h3 6 h4 7 h5 0 h1	2 v 3 a2 4 h2 5 h3 6 h4 7 h5 0 h1	4 h2 5 h3 6 h4 7 h5 0 h1 1 a1 2 v	0 h1 1 a1 2 v 3 a2 4 h2 5 h3 6 h4 7 h5
		6 [126,720]		10 [211,200]	12 [253,440]	10 [211,200]	14 [295,680]	10 [211,200]	14 [295,680]	14 [295,680]	16 [337,920]
slope profile data	slope	4 [84,480]	4 [84,480]	8 [168,960]	8 [168,960]	4 [84,480]	8 [168,960]	0 [0]	4 [84,480]	4 [84,480]	8 [168,960]
Rut data	speed	4	4	4	4	4	4	0	4	4	4
	ruts	0	0	0	4	4	8	12	12	12	12
	roughness	4	4	8	8	4	8	0	4	4	8
	total	8 [16,896]	8 [16,896]	12 [25,344]	16 [33,792]	12 [25,344]	20 [42,240]	12 [25,344]	20 [42,240]	20 [42,240]	24 [50,688]
Plotting Data	Elevation benchmarks	4 [8,448]	4 [8,448]	8 [16,896]	8 [16,896]	4 8,448]	8 [16,896]	0 [0]	4 [8,448]	4 [8,448]	8 [16,896]
extra space on tape		13%	13%	0%	13%	44%	23%	88%	49%	49%	30%

Numbers in brackets are bytes/mile. Others are bytes/sample. Numbers for Rut and Plot data assume 10:1 decimation

Figure 20. Tape space requirements for ten transducer configurations.

points at the end of the file may be ignored during processing. Figure 21 shows the relationships between sample number and the distance traveled for the various forms of data. The figure shows a consistent convention for relating sample number to distance, in which the first sample in the raw data file is defined as occurring 1/2 of the sample interval before the start of the test site.

Processed Data files

When the raw transducer signals are converted to profiles of slope, elevation, roughness, and rut depth, the original data portion of the file is overwritten. The main reason for doing this is to minimize the need for fast-forwarding and rewinding the tape during processing. A second reason is to avoid using excessive tape space copying files. After the processing is complete, the file will contain three distinctly different types of data. One type is the longitudinal slope profile of one or two wheeltracks, sampled at the same rate as the raw data. The second type includes several profiles that are also calculated during processing which do not need to be sampled at such a close interval. These are: rut depth, accumulated roughness, and measurement speed. These are calculated over an interval of DXTRIM = 10 × DELTAX, where DELTAX is the sample interval used for the raw data. The third type includes elevation profiles that are stored at intervals of DXTRIM, which are used to provide quick plots of profile and to ensure that detailed plots overlay properly.

The raw data from the digitizer require two bytes for each number stored; after processing, the data require four bytes for each number. Figure 20 shows the space required to store each kind of data, based on a nominal sample interval of 3 inches. Note that in configuration no. 3 (two profiles, no rut depth) the processed data take up exactly the same amount of tape space as the raw data. This is why a decimation ratio of 10:1 was chosen for the rut and elevation data.

The tape files may hold more data than will fit into the memory of the computer. When this occurs, the file is processed using buffers and the three types of data are interleaved as shown in figure 18 from the section *Computation Methods*. The first buffer starts immediately after the header of the file. Each buffer has a length of NBUFFW reals. (Since all of the processed data are real*4 numbers, it is convenient to use the size of a real*4 number—4 bytes—as a unit of length. Using this convention, the total length in bytes is NBUFFW × 4). In each buffer, the first NPRFFW reals contain the slope profile(s). The next NRUTFW reals contain the rut (and roughness and speed) profile(s), and the following NELVFW reals contain the plotting elevation profile(s). The remaining part of each buffer is not used. (These sizes are added to the header of the file, along with the number of channels in each of the three data sections. The channel numbers of the various profiles are also put into the header section of the file.) The final buffer in the file will usually contain less data than the others, but the buffer size is the same. When more than one buffer is needed, the file is made large enough to hold one complete extra buffer

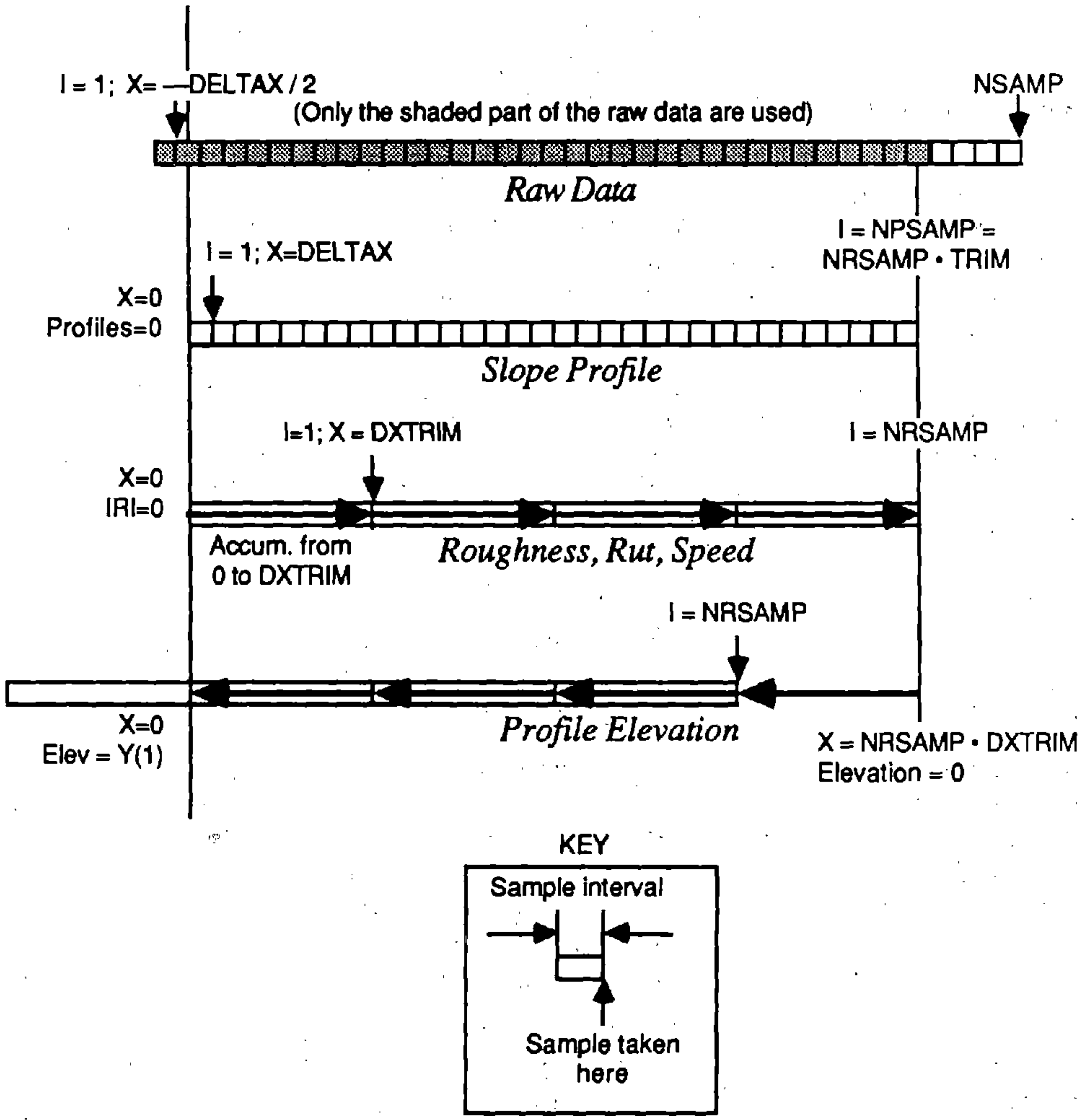


Figure 21. Relation between longitudinal distance and sample number.

when the measuring ends. (This is why there is sometimes a delay from the time that a test is ended to the time that the system finishes writing to tape.)

The writing of the data into this interleaved form is performed only once, by the PRFCMP subroutine that controls all of the data preprocessing. All reading of the data after this is performed by the subroutine RDTAPD.

The slope profiles have units of slope, as defined by the units used for the height sensors and the sample interval. These units cannot be changed by the operator, but could be changed in the future by making a minor alteration in the software. (The scaling is defined by names of the units and several scale factors stored in the header of the file.) The units now used are in/ft. The sample interval is stored in the header as the Fortran variable DELTAX.

The computation method used for the slope profiles is designed to provide the greatest amount of information possible. Whenever an accelerometer is integrated for a long time, it is necessary to remove the lowest frequencies (longest wavelengths) because the noise in the accelerometer is more significant than the acceleration from the road. The PRFCMP subroutine used for this software sets the cut-off wavelength as a function of the test speed, so that at higher speeds the additional information available for long wavelengths is retained.

The first slope value is at the start of the test, at position x=0. (It is the slope from –DELTAX/2 to +DELTAX/2.) The final value is number NPSAMP, and the length of the test is DELTAX × NPSAMP. (See figure 21.)

The "rut" part of the file contains three kinds of signals: rut depth, test speed, and roughness. All three signals are calculated for every sample, but they are then averaged over ten samples and only the averages are stored. The sample interval for these signals is stored in the header as the Fortran variable DXTRIM. The rut depth signals have the same units as the height sensors. Presently, the units are inches. The speed signal has units of mi/h. The first sample for these signals is the average from x=0 to x=DXTRIM. The final sample is the average over the interval x=(NRSAMP - 1) × DXTRIM to x=NRSAMP × DXTRIM. (See figure 21.)

The roughness signals stored in the rut part of the data files are actually accumulated roughness, with the same units used for the height sensors (presently inches). Roughness is always a positive quantity, and therefore the accumulated roughness always increases from the beginning of the file to the end. The roughness between two points is obtained by taking the difference in the accumulated roughness at each point and dividing by the distance between points. (This would give units of in/ft, so an additional scale factor of 5280 ft/mi, contained in the header, is used to show roughness with units of in/mi.) The accumulated roughness by definition begins with zero roughness at x=0. The first value in the roughness part of the file is the accumulation from x=0 to x=DXTRIM, and the last value is the accumulation obtained by the end of the run, where x=NRSAMP × DXTRIM. (See figure 21.)

The third part of the file contains profile elevations that are needed for the plotting software. They have the same units as the height sensors, presently inches. The elevation profiles are calculated for every sample taken, but only one out of every ten values is stored in the file. The interval between samples is stored in the header as the Fortran variable DXTRIM. These profiles are computed with the minimum filtering that can be used for the test speed. The appearance of an elevation profile is strongly dependent on the cutoff wavelength used during the profile computation. Thus stored elevation profiles obtained at different speeds will look different, because the filtering retains the additional long wavelength information obtained with the higher speeds. The profilometer software applies additional filtering when showing plots, such that profiles obtained at different speeds will appear identical if the same filter baselength is selected by the user. (The only effect of the measuring speed that is shown to the user is the fact that longer baselengths are permitted when tests are made at high speeds.)

Because the elevation is obtained using a backwards integration, the mapping between sample number and distance is different than with the rut depth, roughness, and speed signals. They all omit a value for x=0, and begin with an average taken from x=0 to x=DXTRIM. In contrast, the elevation file includes a value for x=0, but omits the value for x=NRSAMP × DXTRIM as indicated in figure 21. (The arrows in the figure show the direction used in processing the data.) By definition, the elevation has a value of 0 at the end of the file, at x=NRSAMP × DXTRIM, and therefore that point is not needed in the file. The subroutine GETELV provides the extra elevation value of zero when an elevation at the end of the run is needed by the plotting software.

PROFILOMETER SUBROUTINES

This section describes the library of subroutines that makes up the profilometer software. The subroutines may be used by programmers wishing to further develop the system, or to adapt some of the profilometer software to other applications. It also lists and describes the individual files that are executable or are referenced when using the profilometer software.

The software makes use of two additional libraries. One of these is a commercial product called Halo, which contains subroutines for controlling graphic elements on the screen during plotting. These subroutines are described in the Halo documentation. The other library, developed by M. Hagan (before this project), extends the Fortran language to provide the control of the screen needed to allow user friendly interaction with the software. The subroutines in this library are described in appendix C.

Most of the software is written in the Fortran language in the form of numerous subroutines. Table 2 lists these subroutines in alphabetical order and provides a quick reference. These routines are described in the remainder of this section, grouped by type in the various subsections. The source listings for those subroutines are included in appendix D. Many of these subroutines use Fortran common blocks to share information. The common blocks are defined in *include* files, described below after the conventions used in this section are defined.

Conventions

File Names

The profilometer software consists of a single executable file called *PROFILE.EXE* and several supporting text files. These are normally stored on the bubble memory of the profilometer system, and are listed in the next subsection. The PROFILE.EXE file is created by compiling the Fortran code and linking the resulting object files together with the appropriate libraries.

The files that are provided have MS-DOS extensions, following the conventions:

— (no extension) text files used by PROFILE or include files required to compile some of the Fortran subroutines.

.BAS — file containing code in the Basic language. (CNTTST.BAS is the only Basic file.)

.EXE — executable file. (PROFILE.EXE is the only executable file.)

.FOR — Fortran source files. If the source file exists, it will have the same name as the corresponding .OBJ file. (For example, the file RDTAPD.FOR

Table 2. List of all Fortran subroutines used with the profilometor.

ACAL (ICHAN, ROW) — Calibrate an analog data channel.
ADCHECK — Check the calibration of th A/D and D/A converters.
ADSET (ADCURB, BUFT, NBUF, BYTB, MAXB, BUFFCNT, DONE)—Set up the data collection parameters and the interrupt routine.
A2DONE (ICH, IGAIN, FREQ, NSAMPS, AV, VNSE) — Collect A/D on channel ICH.
AVEVEL (IBUF, NC1, NS, RBUF, NC2, TRIM, GAIN, BIAS) — Average and decimate a (speed) signal.
BATCH (DR) — Process a list of data files.
CALDA (V) — Set calibration D/A.
CALIB — Calibrate the analog hardware and check the height sensors.
CALREL (ICH, ION) — Switch calibration relay.
CHKSAT (HANDLE, AUTO) — Check the raw transducer signals for saturation.
CONFIGURE — Select which data to collect.
DEBIAS (ARRAY, NCH, NS, BIAS) — Subtract bias from signal in real*4 array.
DTCLEAR — Clear the Data Translation board.
DTCLOCK (F) — Set the A/D clock on the Data Translation board.
FILCLK (F) — Set the filter clock
GETELV (SKPLOT, NSMP, MOVAV1, MOVAV2, QNDPLT, HANDLE, IERR) — Get elevation profiles from tape.
GETLEN (X, XLL, XUL, UNITS, TITLE, PROMPT, IRET) — Prompt the user for some type of length measure or range.
GOAHED (HANDLE) — Warn the user that some processing needs to be done.
GRCURS (ISTART, IPLT, KCURS, NPTS, IMAX, NPTOT, NPMAX,IUPDT, XMIN, XMAX, XSTART, DX, YMIN, YMAX, ICH) — Wait for the user to hit a key, then update plot parameters.
HIPASS (ARRAY, NCH, N1, N2, N3, N4, N5, MOVAV1, MOVAV2) — Filter a signal with a hipass filter.
Function IAVE (ARRAY, NCH, NS) — Average value of signal in integer*2 array.
INITIO — Initialize I/O.
INITP — Initialize status variables and check the A/D board and the floating point processor.
IOEX — Present a menu of options to exercise the input/output hardware.
LABEL (X, STRING, L) — Convert a real number into a string for Halo.
LOADTP — Load and initialize tape.
LOGO — Draw the logo for the profilometer.
LOPASS (ARRAY, NCH, NS, MOVAV1, MOVAV2) — Smooth a signal.
LRSLOP (ARRAY, NDIM, NSAMP, SLOPE) — Calculate slope of signal using a linear regression.
MAIN — Show the Logo and offer the main menu to the user.
MEASURE — Generate the menu for measuring data.

Table 2. List of all Fortran subroutines used with the profilometer (continued).

MINV (ARRAY, N, D, LARRAY, MARRAY) — Matrix inversion.
PLOT (MODE, IACTIV, NCHAN, NPTS, ICH, IIS, ITOT, DX, XMIN, XMAX, XSTART, KCURS, YMIN, YMAX, NAME, UNITS, XNAME, XUNITS, GAIN, OFF, IUPDT, ISTART, NPTOT, NPMAX, TITLE) — Plot data using Halo subroutines.
PLTELV (HANDLE, QNDPLT) — Set up plots of profile elevation.
PLTRAW (HANDLE) — Set up plots of raw signals.
PLTRUT (HANDLE) — Set up plots of rut-depth and roughness signals.
PLTSEL (NCHAN, NAME, UNITS, XNAME, XUNITS, DX, XMIN, XSTART, XRANGE, YRANGE, YMXRNG, NPTS, NPMAX, NPTOT, KCURS, ICH) — Prompt user for the selection of channels and plotting ranges.
PRFCMP (HANDLE) — Convert raw data into slope profile, rut depth, IRI roughness, and elevation profile.
PRFELV (BUF1, NC1, NS, BUF2, NC2, TRIM, DX, C, ENDELV) — Compute compressed elevation profile from slope.
PRFIRI (BUF1, BUF2, X1, X2, X3, X4, ROUGH) — Filter a slope profile signal using the IRI quarter-car simulation.
PROCESS — Generate the menu for viewing data and call the appropriate subroutines.
PRTLF (LSCR, LLPT, LFL) — Add carriage returns after each line.
PRTNUM (HANDLE) — Print numerics averaged over a specified interval.
PULSE — Check the calibration of the distance sensor.
PULTST (PASS, DONE, JJ, CONV, MAXP) — Set up the interrupt and data collection routine for the distance pulser check.
PUTYN (YESNO, IROW, ICOL) — Put Y or N in specified screen location.
Function RAVE (ARRAY, NCH, NS) — Average value of signal in real*4 array.
RDSET — Read in SETUP array from a text file.
RDTAPD (HANDLE, ARRAY, WHICH, OFFSET, NSMP, IERR) — Read numerical data from processed file.
RDTAPE (HANDLE, ARRAY, OFFSET, NBYTES, IER) — Read binary data.
RESTOR — Restore analog signal conditioning unit.
RUTCMP (HL, HC, HR, NCHRAW, NS, RUT, NCHRUT, TRIM, GAINL, GAINC, GAINR, ZL, ZC, ZR, HLLAT, HRLAT) — Compute, average, and decimate a rut-depth signal.
SATMAX (ARRAY, NCH, NS, OFFSET, MAX, COUNT, NSAT, LSAT) — Check raw data signal for saturation at upper limit.
SATMIN (ARRAY, NCH, NS, OFFSET, MIN, COUNT, NSAT, LSAT) — Check raw data signal for saturation at lower limit.
SCLDWN (X, XNORM, XDOWN) — Scale a variable down.
SCLUP (X, XNORM, XUP) — Scale a variable up.

Table 2. List of all Fortran subroutines used with the profilometer (continued).

SETAD (AD) — Set up the A/D parameters on the Data Translation board.
SETDMA (DM) — Set up the DMA controller.
SETSTM — Calculate coefficients for quarter-car simulation.
SETUPS — Edit the transducer information.
STARTAD (FF,BUFST,BUFT,BUFFCNT,MAXB,ADCURB,DONE,INDEX) — Start the data collection.
TCHECK (IC,ROW,IPOS) — Check a height transducer.
TEST (IITY)—Collect data.
TIKSET (XMIN, XMAX, TICK, TMIN, TMAX, NTICK) — Determine first and last tick marks in a given range.
TSTDIS — Display summary of test parameters.
TWAIT (T)—Wait for a time interval.
UNLDTP—Unload the tape.
UPDSET (HANDLE) — Update the SETUP array that begins the current data file.
WRTAPE (HANDLE, ARRAY, OFFSET, NBYTES, IER) — Write binary data.
WRTSCR (FNAME) — Read names and coordinates from file, create screen display.
WRTSET — Write the SETUP array to a text file.
YESNOL (YESNO,IROW,ICOL,IRET) — Get Yes/No answer and set logical variable.
ZOFF (ICH,OFFSET) — Set the offset on an analog card.

contains the source code for the subroutine RDTAPD and the file RDTAPD.OBJ contains the compiled subroutine.)

.LIB — libraries of subroutines that can be linked to other software.

.OBJ — object files. These files contain one or more subroutines that have been compiled and which can be linked to other software using the MS-DOS linker.

Subroutine Descriptions

The subroutines are documented in the following subsections. Each subsection covers a category, and the subroutines within that category are listed in alphabetical order. The first line in each description gives the name of the subroutine, the argument(s) for the subroutine in parentheses, and the name of the object file. If a Fortran source listing exists, it will be in a file with the same name as the object file, but with the .FOR extension rather than the .OBJ extension of the object file. Next, the procedure performed by the subroutine is described. The arguments are then listed in the order in which they appear when calling the subroutine. Symbols are used to designate whether an argument is an input or an output:

→ the argument is an input and is never modified by the subroutine. Constants can be used for these arguments. If variables are used, they must have values before the subroutine is invoked.

← the argument is an output and is set by the subroutine. Constants must not be used for these arguments. Variables need not be initialized before calling the subroutine.

↔ the argument is both an input and an output. The subroutine uses the initial value of the variable, but may update it. Constants must not be used for these arguments. Variables must be initialized before calling the subroutine.

Finally, any *include* files needed to compile the subroutine (using the source listing) will be cited. The actual *include* statements used in the Microsoft Fortran compiler are shown. A short discussion of why the file must be included is usually provided.

Common Blocks and Auxiliary Files

Files Used by the Profilometer Software

Several files are accessed by the program PROFILE, and are listed in table 3. The contents of the text files are included in appendix D along with the source listings. All but the file NAME.VOL should be present in the bubble memory (drive C) in order for the software to function properly. The following descriptions of the individual subroutines specify whether an auxiliary file is used by that subroutine.

Table 3. List of auxiliary files needed by the profilometer software.

File Name	*Description*
CONFIG.SET	text file containing parameters that define the ten possible transducer configurations.
LOGO	text file containing screen coordinates and labels used for the logo.
NAME.VOL	file on the tape with variables describing the tape status.
PRTSCR	text file containing screen coordinates and labels used for setting up to print numerics.
SETUP.SET	binary file with current setup data, equivalenced to the SETUP array.
TSTSCR	text file with screen coordinates and labels used for creating the screen display for a test setup.

Include Files

To aid in the development and maintenance of the software, the lines of code that define these blocks are kept in special *include* files. When a program is compiled, an *include* file is inserted into the program in place of an *include* command that gives the name of the file. When many subroutines employ the same code, that code can be put into an *include* file to shorten the files for the individual programs. If the code in the *include* file is modified, the various files that make reference to the *include* file need not be changed. (However, they must be recompiled.) Many of the profilometer subroutines share data using Fortran common blocks. The definitions of these blocks and the variables they contain are kept in include files.

Table 4 shows which subroutines make use of five include files used in the development of the software, described below:

- *BUFCOM* — defines the common block BUFFER, which contains 262,144 bytes of memory that are used to store samples of variables that are measured and processed at various times. The memory can be addressed using three arrays that overlay the same space through use of the Fortran EQUIVALENCE statement. These arrays are:

  IBUF — an integer*2 array of length 131,072.

  PCBUFR — a real*4 array of length 65,536.

  PCBUFI — an integer*2 array of length 131,068 which has an offset of 8 bytes in the EQUIVALENCE, needed for the profile computation subroutine.

  The size of the array PCBUFR is also available as the Fortran parameter MXBFSZ.

- *HANDLES* — defines several variables that are listed in table 5.

- *IOPARMS* — defines Fortran parameters that are required when accessing the I/O hardware. These parameters are listed in table 6.

- *SETCOM* — defines an integer*2 array SET in the common block SETCOM. Every data file measured for a road test or bounce test begins with 2048 bytes that correspond to this, which overlays a number of smaller arrays and scalar variables by using Fortran equivalence statements. Table 7 lists all of the variables that are contained in this common block, and table 8 shows how these variables are mapped onto the SET array through the use of equivalence statements.

  This is where any data related to a test is kept, other than the sampled values of the test variables. Some of these variables are set before the test (time, number of channels, etc.). The number of samples is defined at the completion of a test. Other variables are set during the various stages of data processing. Finally, some variables are used to record how the data are plotted, so that the next time the plotter is invoked the default values will be those most recently selected by the user.

Table 4. Map showing the usage of *include* files.

	BUFCOM	HANDLES	IOPARMS	SETCOM	STATCOM
ACAL				X	
ADCHECK			X		
ADSET					
A2DONE	X		X		
AVEVEL					
BATCH		X		X	
CALDA			X		
CALIB				X	X
CALREL			X		
CHKSAT	X	X		X	
CONFIGURE				X	
DEBIAS					
DTCLEAR			X		
DTCLOCK			X		
FILCLK			X		
GETELV	X			X	
GETLEN					
GOAHED				X	
GRCURS					
HIPASS					
IAVE					
INITIO			X		
INITP			X	X	X

Table 4. Map showing the usage of *include* files (continued).

	BUFCOM	HANDLES	IOPARMS	SETCOM	STATCOM
IOEX			X		
LABEL					
LOADTP				X	X
LOGO					
LOPASS					
LRSLOP					
MAIN	X			X	X
MEASURE					X
MINV					
PLOT	X				
PLTELV	X	X		X	X
PLTRAW	X	X		X	X
PLTRUT	X			X	X
PLTSEL					
PRFCMP	X			X	
PRFELV					
PRFIRI				X	
PROCESS		X		X	X
PRTLF					
PRTNUM	X			X	X
PULSE	X		X	X	
PULTST					
PUTYN					

Table 4. Map showing the usage of *include* files (continued).

	BUFCOM	HANDLES	IOPARMS	SETCOM	STATCOM
RAVE					
RDSET				X	
RDTAPD				X	
RDTAPE		X			
RESTOR			X	X	
RUTCMP					
SCLDWN					
SCLUP					
SETAD			X		
SETDMA					
SETSTM				X	
SETUPS				X	
STARTAD	X		X	X	
TCHECK				X	
TEST	X	X	X	X	X
TIKSET					
TSTDIS				X	X
TWAIT					
UNLDTP				X	X
UPDSET				X	
WRTAPE		X			
WRTSET				X	
YESNOL					
ZOFF			X		

Table 5. Variables from the HANDLES *include* file.

Name	*Type*	*Definition*
ACCESS	integer*2	access method for file opening (0=read only, 1=write, 2=read and write.
BYTES	integer*4	number of bytes to read or write.
HANDLE	integer*2	file handle assigned by DOS.
METHOD	integer*2	Method of file positioning (0=absolute, 1=relative, 2=from the end).
OFFSET	integer*4	offset into a file.
POINTER	integer*4	returned file pointer.
RBYTES	integer*4	actual number of bytes read or written.

Table 6. Definitions of the I/O parameters from the IOPARMS *include* file.

Parameter	*Value*	*Definition*
AADDR	#305	address of analog address lines
ADATA	#300	address of analog data lines
CCLEAR	1	clear command
CCLOCK	3	set A/D clock command
CDA	8	D/A command
CNTRL	#307	address of PIO control register
CROFF	#0C	value to turn on cal relay
CRON	#0D	value to turn off cal relay
CSAD	#0D	command to setup A/D parameters
CSTOP	15	A/D stop command
CWAIT	4	mask for command completion test
DADIS	#04	value to disable D/A relay
DAEN	#05	value to enable calibration D/A relay
DASOFF	#0B	value to turn off D/A strobe
DASON	#0A	value to turn on D/A strobe
DTCOM	#2ED	data Translation board command address
DTDATA	#2EC	data Translation board data address
DTSTAT	#2ED	data Translation board status address
INTD	#310	interrupt disable address
INTE	#30C	interrupt enable address
IPC	#306	port C address of PIO
RWAIT	5	mask for read completion test
SHOFF	#08	value to turn off shunt cal relay
SHON	#09	value to turn on shunt cal relay
TIMERC	#309	9513 timer control address
TIMERD	#308	9513 timer data address
WWAIT	2	mask for write completion test
ZAEN	#0F	value to enable card D/A's
ZDIS	#0E	value to disable card D/A's

# denotes hexadecimal number

Table 7. Variables stored in the SETCOM common block and *include* file.

Following are the variables that are equivalenced into the array SET which occupies the first 2048 bytes of each test file.

Variable	*Type*	*Definition*
ADSTOP	integer*2	ending A/D channel in a test scan.
ADSTRT	integer*2	starting A/D channel in a test scan.
AMPGA	real*4	array for actual amplifier gains.
AMPGN	real*4	array for nominal amplifier gains.
AVEBAS	real*4	baselength last used for smoothing plots.
CHID	character*8	array for channel names. 1-8=transducers,9=interval between pulses, 10=distance, 11=IRI.
CMT	character*64	comment.
COFINT	real*4	coefficient for integration.
CRUT	log*4	.true. if there is a center rut signal.
DELTAX	real*4	sample interval for raw data.
DIRECT	character*8	direction traveled.
DXTRIM	real*4	sample interval for decimated data (m).
FLTBAS	real*4	baselength last used to remove long waves.
GAIN	real*4	array for channel gains.
H?LAT	real*4	distances between height sensors (?=1,2,4,5).
ICH?	integer*2	channel ID (offset) for sensors (?=A1,A2,H1,H2,H3,H4,H5,V).
ICR	integer*2	channel ID (offset) for center rut signal.
IDAY	integer*2	day for time-of-test.
IDMODE	integer*2	dode for test counter register.
IDIV	integer*2	value for counter.
IH	integer*2	hour for time-of-test.
IM	integer*2	month for time-of-test.
ILIRI	integer*2	channel ID (offset) for left-hand IRI roughness.
ILPRF	integer*2	channel ID (offset) for left-hand profile.
ILR	integer*2	channel ID (offset) for left-hand rut signal.
IMIN	integer*2	minute for time-of-test.
IOFFS	integer*2	array of offsets stored in the channel offset D/A.
IRIRI	integer*2	channel ID (offset) for right-hand IRI roughness.
IRPRF	integer*2	channel ID (offset) for right-hand profile.
IRR	integer*2	channel ID (offset) for right-hand rut signal.
ISEC	integer*2	second for time-of-test.

Table 7. Variables stored in the SETCOM common block and *include* file — continued.

Variable	*Type*	*Definition*
ITSOK	logic	used after chksat and before PRFCMP.
IVEL	integer*2	channel ID (offset) for decimated velocity signal.
IYR	integer*2	year for time-of-test.
LANE	character*12	name of test lane.
LGLTCH	integer*4	array with locations the raw signals first glitched.
LNGWAV	real*4	longest wavelength of interest when integrating during profile computation.
LPROF	log*4	.true. if there is a left profile signal.
LRUT	log*4	.true. if there is a left rut signal.
LSAT	integer*4	array with locations the raw signals first saturated.
MAXBUF	integer*4	maximum number of (4-byte) words available for signal processing (profile computation, etc.).
MAXLEN	real*4	maximum test length.
NBUFFW	integer*4	number of full-words between starts of buffers on tape after processing.
NBUFS	integer*2	number of buffers in tape file after processing.
NCHAN	integer*2	number of raw data channels.
NCHPRF	integer*4	number of profiles.
NCHRAW	integer*4	number of raw data channels.
NCHRUT	integer*4	number of channels in compressed rut file.
NELVFW	integer*4	number of full-words/buffer in elevation "file."
NGLTCH	integer*4	array with number of times the raw signal glitched.
NPRFFW	integer*4	number of full-words/buffer in slope profile "file."
NPSAMP	integer*4	number of samples/buffer for slope profile.
NRSAMP	integer*4	number of samples/buffer for rut & elevation.
NRUTFW	integer*4	number of full-words/buffer in rut "file."
NSAMP	integer*4	total number of samples of raw data (same as PASSA).
NSAT	integer*4	array with number times the raw signals saturated.
NSPTOT	integer*4	total number of samples of slope profile.
NSRTOT	integer*4	total number of samples of rut & elevation.
OFFS	real*4	array for physical offsets.
OPER	character*16	name of test operator.
PASSA	integer*4	actual number of A/D scans (i.e., points/channel).
PINC	real*4	print interval.
PRM	real*4	array of 4 coefficients used for 1/4 car.
PSTART	real*4	starting print position.
PSTOP	real*4	stopping print position.
RAWIX	integer*4	not used in this version.
ROUTE	character*16	name of route.

Table 7. Variables stored in the SETCOM common block and *include* file — continued.

Variable	*Type*	*Definition*
RPROF	log*4	.true. if there is a right profile signal.
RRUT	log*4	.true. if there is a right rut signal.
CSAMP	real*4	number of samples per foot.
SCLFA	real*4	scale factor needed to convert acceleration to m/s/s.
SCLFDX	real*4	scale factor needed to convert DELTAX to m.
SCLFH	real*4	scale factor needed to convert height to m.
SCLFRI	real*4	scale factor for roughness. Units would normally be height/DELTAX; this is added to get in/mi. (for US units, SCLFRI = 5280.)
SCLFV	real*4	scale factor needed to convert speed to m/s.
STM	real*4	array of 16 coefficients used for 1/4 car.
SURF	character*16	type of road surface.
TFILE	character*16	test file name.
TCONFI	integer*2	test configuration number (1 - 10 valid values).
TRIM	integer*4	decimation ratio for rut and elevation data.
TSTCON	character*32	test configuration name (e.g. 'Left Profile').
TSTTYP	integer*2	status of data file. 0=raw test, 1=bounce, 2=processed, 3=raw after check for saturation. 4=speed signal too low, can't process. 5=bounce test that has been checked. 6=bounce test that has been processed. 7=file was ruined during processing.
TSTSPD	integer*2	nominal test speed in mi/h.
UNITS	character*8	array for channel units.
XCURS	real*4	cursor position last used in plotting.
XDUCGN	real*4	array for transducer gains.
XDUCT	integer*2	array of transducer types: 0,1, or 2
XRANGE	real*4	range covered in last plot.
VELMAX	real*4	maximum speed found during test.
VELMIN	real*4	minimum speed found during test.
ZDATA	real*4	array of zero data values. Convert integer values into engineering units with the equation: REAL = GAIN(I) * float(INTEGER) - ZDATA(I)

Table 8. Equivalences used for the SET variables.

SET(1)-(32)	GAIN	SET(509)	ILR
SET(33)-(64)	ZDATA	SET(510)	ICR
SET(65)-(96)	AMPGN	SET(511)	IRR
SET(97)-(160)	CHID	SET(512)-(526)	RESERVED
SET(161)-(224)	UNITS	SET(527)-(528)	NCHRAW
SET(225)-(240)	IOFFS	SET(529)-(530)	NCHPRF
SET(241)-(256)	XDUCT	SET(531)-(532)	NCHRUT
SET(257)-(288)	XDUCGN	SET(533)-(548)	RESERVED
SET(289)-(320)	AMPGA	SET(549)-(550)	LPROF
SET(321)-(384)	OFFS	SET(551)-(552)	RPROF
SET(385)-(416)	CMT	SET(553)-(554)	LRUT
SET(417)	IYR	SET(555)-(556)	CRUT
SET(418)	IM	SET(557)-(558)	RRUT
SET(419)	IDAY	SET(559)-(590)	STM
SET(420)	IH	SET(591)-(598)	PRM
SET(421)	IMIN	SET(599)-(600)	NPSAMP
SET(422)	ISEC	SET(601)-(602)	NRSAMP
SET(423)	TCONFI	SET(603)-(604)	NBUFFW
SET(424)	TSTSPD	SET(605)-(606)	NRUTFW
SET(425)-(426)	SAMP	SET(607)-(608)	NPRFFW
SET(427)	IDMODE	SET(609)-(610)	NELVFW
SET(428)	IDIV	SET(611)-(612)	NSPTOT
SET(429)-(430)	PASSA	SET(613)-(614)	NSRTOT
SET(431)-(432)	MAXLEN	SET(615)-(616)	DELTAX
SET(433)-(436)	DIRECT	SET(617)-(618)	DXTRIM
SET(437)-(444)	ROUTE	SET(619)-(620)	TRIM
SET(445)-(452)	OPER	SET(621)-(622)	LNGWAV
SET(453)-(460)	SURF	SET(623)-(624)	COFINT
SET(461)-(466)	LANE	SET(625)-(626)	MAXBUF
SET(467)-(474)	TFILE	SET(627)	NBUFS
SET(475)-(490)	TSTCON	SET(629)-(630)	NSAMP
SET(491)-(492)	RAWIX	SET(631)	TSTTYP
SET(493)	NCHAN	SET(633)-(634)	SCLFA
SET(494)	ADSTRT	SET(635)-(636)	SCLFDX
SET(495)	ADSTOP	SET(637)-(638)	SCLFH
SET(496)	ICHH1	SET(639)-(640)	SCLFV
SET(497)	ICHA1	SET(641)-(642)	H1LAT
SET(498)	ICHV	SET(643)-(644)	H2LAT
SET(499)	ICHA2	SET(645)-(646)	H4LAT
SET(500)	ICHH2	SET(647)-(648)	H5LAT
SET(501)	ICHH3	SET(649)-(664)	LSAT
SET(502)	ICHH4	SET(665)-(680)	NSAT
SET(503)	ICHH5	SET(681)-(696)	LGLTCH
SET(504)	ILPRF	SET(697)-(712)	NGLTCH
SET(505)	IRPRF	SET(713)-(714)	VELMIN
SET(506)	ILIRI	SET(715)-(716)	VELMAX
SET(507)	IRIRI	SET(717)-(718)	SCLFRI
SET(508)	IVEL		

- *STATCOM* — defines several variables that describe the status of the system, which are equivalenced to arrays STAT and TVOL in the unlabelled common block. Table 9 defines these variables.

File Access

This subsection describes the subroutines that deal with the binary data files that contain the measures from the profilometer.

Quick Reference

RDSET — Read in SETUP array from a binary file.
RDTAPD (HANDLE, ARRAY, WHICH, OFFSET, NSMP, IERR) — Read numerical data from processed file.
RDTAPE (HANDLE, ARRAY, OFFSET, NBYTES, IER) — Read binary data.
TSTDIS — Display summary of test parameters.
UPDSET (HANDLE) — Update the SETUP array that begins the current data file.
WRTAPE (HANDLE, ARRAY, OFFSET, NBYTES, IER) — Write binary data.
WRTSET — Write the SETUP array to a binary file.

Subroutine Descriptions

RDSET *rdwrtset.obj*

Read in SETUP array from drive C (bubble). The name of the file is SETUP.SET.

$INCLUDE: 'SETCOM' The SETUP array is contained in a common block.

RDTAPD (HANDLE, ARRAY, WHICH, OFFSET, NSMP, IERR) *rdtapd.obj*

This subroutine reads numerical data from tape. It allows the calling program to treat the data on tape as if it were contiguous, instead of the interleaved format that is actually used.

→	HANDLE	integer*2	handle for tape file.
←	ARRAY	real*4	array in memory that holds the data read from the tape.
→	WHICH	integer*2	code for data type. 1=slope profile, 2=rut stuff, 3=profile elevation.
→	OFFSET	integer*4	number of samples to skip before 1st.
↔	NSMP	integer*4	number of samples to read. If NSMP is too large and goes beyond the range of data existing on tape, the subroutine will reset NSMP to the number of samples actually read.
→	IERR	integer*2	error return code. 0=cool.

Table 9. Variables from the STATCOM *include* file.

Name	*Type*	*Definition*
BOUNYN	integer*2	this is a 1 if a bounce test has been done since power up.
CALCON	integer*2	configuration number at the time of the last calibration.
CALTIM	integer*2	time in seconds of the last calibration.
CALYN	integer*2	this is 1 if a calibration has been done since power up.
FINIT	integer*2	this is a 1 if a file has been selected for processing.
LFILE	character*16	the name of the last test file.
PFILE	character*16	the name of the file being processed.
TCDATE	character*8	date when the tape was created.
TCTIME	character*8	time when the tape was created
TINIT	integer*2	this is a 1 if a tape has been loaded.
TLDATE	character*8	date when the last file was created.
TLTIME	character*8	time of the last file creation.
TNAME	character*8	name assigned to the tape.

$INCLUDE: 'SETCOM' the subroutine uses the variables in this common block that describe the layout of the processed data in the file. These include the number of channels in each sector, the number of points, full-words, etc.

RDTAPE (HANDLE, ARRAY, OFFSET, NBYTES, IER) *rdwrtape.obj*

Read binary data from tape or disk file.

→	HANDLE	integer*2	file handle.
←	ARRAY	integer*2	destination array for data.
→	OFFSET	integer*4	Offset into file 0=start.
→	NBYTES	integer*4	number of bytes to read.
←	IER	integer*2	error return 0=no error.

$INCLUDE:'HANDLES' includes a few integer*4 variables.

TSTDIS *tstdis.obj*

Display information about the data in a file using information from the SETCOM header block. If PASSA < 1, file is taken from TFILE and the bottom half of the screen is left blank. If PASSA > 0, then it is an existing data file and extra information is shown in the bottom 1/2 of the screen. This subroutine requires a file named TSTSCR. that contains the screen coordinates of the stuff that is displayed.

$INCLUDE:'STATCOM' contains the name of the default file.
$INCLUDE:'SETCOM' the SETUP array is contained in a common block.

UPDSET (HANDLE) *rdwrtset.obj*

Update the SETUP array that begins the current data file. This subroutine is used when some of the SETUP variables have been modified in some way.

→ HANDLE integer*2 file handle.

$INCLUDE:'SETCOM' The SETUP array is contained in a common block.

WRTAPE (HANDLE, ARRAY, OFFSET, NBYTES, IER) *rdwrtape.obj*

Write binary data to tape or disk file.

→	HANDLE	integer*2	file handle.
→	ARRAY	integer*2	array containing data.

→ OFFSET integer*4 Offset into file 0=start.
→ NBYTES integer*4 number of bytes to read.
← IER integer*2 error return 0=no error.

$INCLUDE:'HANDLES' includes a few integer*4 variables.

WRTSET *rdwrtset.obj*

Write the SETUP array to a file in drive C (bubble). The name of the file is SETUP.SET.

$INCLUDE:'SETCOM' the SETUP array is contained in a common block.

Initialization

This subsection describes the subroutines that initialize the hardware and software of the profilometer.

Quick Reference

ADCHECK—Check the calibration of the A/D and D/A converters.
INITP—Initialize status variables and check the A/D board and the floating point processor.
LOADTP—Load and initialize tape.
SETUPS—Edit the transducer information.
UNLDTP—Unload the tape.

Subroutine Descriptions

ADCHECK *adcheck.obj*

Check the calibration of the A/D and D/A converters.

INCLUDE:'IOPARMS' contains some parameters needed in switching channel 7 inputs.

First, both inputs of channel 7 on the A/D converter are grounded and sampled for one second. Then, a 2.5-volt reference is switched to the input of channel 7 on the A/D converter and sampled for one second. The first reading (average for a second) is subtracted from the second reading and the difference is printed as the "corrected reference voltage." If the measured value is not within .015 volts of 2.5, then a warning message is printed.

The D/A is checked somewhat differently. The D/A is switched to the inputs of channel 7 of the A/D converter and two voltages (±2.5) are put out and sampled. The difference is then compared to 5.0 volts. If the difference is not within .03 volts of 5 volts then a warning message is printed.

INITP *initp.obj*

This subroutine (1) initializes all status variables and reads in the last setup recorded on the bubble drive, (2) checks the Data Translation board by commanding it to execute a self-test, and (3) exercises the floating point processor to ensure correct operation.

INCLUDE:'IOPARMS'	contains some parameters needed to control the Data Translation board.
INCLUDE:'STATCOM'	contains status variables to be initialized.
INCLUDE:'SETCOM'	contains setup variables read in from bubble.

LOADT *loadtape.obj*

Prepare the tape for data storage. If it is a new tape, then a file called NAME.VOL is created and all tape status and file variables are initialized. If it is an old tape, then the file NAME.VOL is read and the tape name, creation date, and last file name are printed on the screen. The drive part of the name is used to set the default drive for subsequent data storage.

INCLUDE:'STATCOM'	contains tape volume information.
INCLUDE:'SETCOM'	contains the name of the next test drive and file name.

SETUPS *setup.obj*

Write all transducer setup information to the screen and allow editing of it by an expert user. This information includes transducer names, units, types, and gains as well as nominal amplifier gains. In addition, the actual amplifier gains and the full-scale values (as determined by the last calibration) are printed but cannot be edited.

INCLUDE:'SETCOM'	contains all transducer setup variables.

UNLDTP *unloadtp.obj*

This subroutine (1) updates the file NAME.VOL with new volume information, (2) causes all directory buffers to be written to tape, and (3) commands the tape to rewind and be unloadable.

INCLUDE:'STATCOM'	contains tape volume information.
INCLUDE:'SETCOM'	contains the name of the last test drive and file name.

I/O Subroutines

This subsection describes the subroutines that interface the profilometer software to the signal-conditioning unit, the Data Translation A/D board, and the calibration control board.

Quick reference

A2DONE (ICH, IGAIN, FREQ, NSAMPS, AV, VNSE)—Collect A/D on channel ICH.
CALDA (V)—Set calibration D/A.
CALREL (ICH, ION)—Switch calibration relay.
DTCLEAR—Clear the Data Translation board.
DTCLOCK (F)—Set the A/D clock on the Data Translation board.
FILCLK (F)—Set the filter clock.
INITIO—Initialize I/O.
RESTOR—Restore analog signal-conditioning unit.
SETAD (AD)—Set up the A/D parameters on the Data Translation board.
SETDMA (DM)—Set up the DMA controller.
TWAIT (T)—Wait for a time.
ZOFF (ICH, OFFSET)—Set the offset on an analog card.

Subroutine Descriptions

A2DONE (ICH, IGAIN, FREQ, NSAMPS, AV, VNSE) *iosubs.obj*

Sample a channel at a specified frequency for a specified number of samples. Then calculate the average value and the RMS noise.

→	ICH	integer*2	channel to sample $0 \leq ICH \leq 47$.
→	IGAIN	integer*2	gain for A/D (0= gain of 1, 1= gain of 2, 2=gain of 4, 3= gain of 8).
→	FREQ	real*4	sampling frequency.
→	NSAMPS	integer*2	number of samples $0 \leq NSAMPS \leq 32767$.
←	AV	real*4	average voltage.
←	VNSE	real*4	RMS noise.

$INCLUDE: 'BUFCOM' collected data are placed in the array buffer.
$INCLUDE: 'IOPARMS' defines addresses or constants related to the hardware.

CALDA (V) *iosubs.obj*

Set the calibration D/A to V volts. The calibration enable relay is also switched and remains on.

→	V	real*4	voltage of calibration D/A.

$INCLUDE 'IOPARMS' defines addresses or constants related to the hardware.

CALREL (ICH, ION) *iosubs.obj*

Switch the calibration relay on an analog channel on or off.

→ ICH	integer*2	channel number $0 \leq ICH \leq 47$.
→ ION	integer*2	ION=0 is off and ION+1 is on.

$INCLUDE 'IOPARMS' defines addresses or constants related to the hardware.

DTCLEAR *iosubs.obj*

Send the clear command to the Data Translation A/D board.

$INCLUDE 'IOPARMS' defines addresses or constants related to the hardware..

DTCLOCK (F) *iosubs.obj*

Set the A/D clock on the Data Translation A/D board.

→ F	real*4	A/D clock frequency (Hz).

$INCLUDE 'IOPARMS' defines addresses or constants related to the hardware.

FILCLK (F) *iosubs.obj*

Set the cutoff frequency of the Butterworth filters.

→ F	real*4	filter clock frequency (Hz).

$INCLUDE 'IOPARMS' defines addresses or constants related to the hardware.

INITIO *iosubs.obj*

Initialize the I/O stuff: (1) initialize IBM analog interface board, (2) initialize the 8255 chip and set up the strobe lines, (3) do a restore to initialize all analog boards, (4) set up the master mode of the 9513 timer chip, (5) set the mode of counters # 4 and 5, and (6) set the values for counters 4 and 5.

$INCLUDE 'IOPARMS' defines addresses or constants related to the hardware.

RESTOR *iosubs.obj*

Turn off all calibration relays, shunt cal relays, and disable the calibration D/A relay. Set the offset D/As on all the cards to the values recorded in the array IOFFS. This function restores the analog card states after the analog power has been switched off then on (usually for diagnostic tests).

$INCLUDE 'IOPARMS' defines addresses or constants related to the hardware.
$INCLUDE: 'SETCOM' uses the array IOFFS to restore the offsets.

SETAD (AD) *iosubs.obj*

Set up the A/D parameters on the Data Translation board. The gain, start and end channels, and number of conversions are sent to the A/D board.

→ AD	integer*2	array for setup parameters where AD(1)=gain, AD(2)= starting channel, AD(3)= end channel, AD(4)= number of conversions, and AD(5)=?.

$INCLUDE 'IOPARMS' defines addresses or constants related to the hardware.

SETDMA (DM) *iosubs.obj*

Set up the DMA controller for data collection. The starting address, number of bytes to be transferred, and the page number are transferred to the DMA controller on the PC motherboard.

→ DM	integer*2	array for the DMA parameters where DM(1)= low byte of the address, DM(2)= high byte of the address, DM(3)=low byte of the number of transfers -1, DM(4)=high byte of the number of transfers -1, DM(5)= page.

TWAIT (T) *iosubs.obj*

Wait for a time T then return to caller.

→ T	real*4	time to wait.

ZOFF (ICH,OFFSET) *iosubs.obj*

Set the D/A on the analog channel # ICH to OFFSET.

→ ICH	integer*2	channel number $0 \leq ICH \leq 47$.
→ OFFSET	integer*2	value for 8-bit D/A $-128 \leq OFFSET \leq 127$.

$INCLUDE 'IOPARMS' defines addresses or constants related to the hardware.

Plotting

This subsection describes the subroutines that scale and plot data or use the Halo graphics subroutines.

Quick Reference

GETELV (SKPLOT, NSMP, MOVAV1, MOVAV2, QNDPLT, HANDLE, IERR) — Get elevation profiles from tape.

GETLEN (X, XLL, XUL, UNITS, TITLE, PROMPT, IRET) — Prompt the user for some type of length measure or range.

GRCURS (ISTART, IPLT, KCURS, NPTS, IMAX, NPTOT, NPMAX,IUPDT, XMIN, XMAX, XSTART, DX, YMIN, YMAX, ICH) — Wait for the user to hit a key, then update plot parameters.

LABEL (X, STRING, L) — Convert a real number into a string for Halo.

LOGO — Draw the logo for the profilometer.

PLOT (MODE, IACTIV, NCHAN, NPTS, ICH, IIS, ITOT, DX, XMIN, XMAX, XSTART, KCURS, YMIN, YMAX, NAME, UNITS, XNAME, XUNITS, GAIN, OFF, IUPDT, ISTART, NPTOT, NPMAX, TITLE) — Plot data contained in common array using Halo subroutines.

PLTELV (HANDLE, QNDPLT) — Set up plots of profile elevation.

PLTRAW (HANDLE) — Set up plots of raw signals.

PLTRUT (HANDLE) — Set up plots of rut depth and roughness signals.

PLTSEL (NCHAN, NAME, UNITS, XNAME, XUNITS, DX, XMIN, XSTART, XRANGE, YRANGE, YMXRNG, NPTS, NPMAX, NPTOT, KCURS, ICH) — Prompt user for the selection of channels and plotting ranges.

SCLDWN (X, XNORM, XDOWN) — Scale a variable down.

SCLUP (X, XNORM, XUP) — Scale a variable up.

TIKSET (XMIN, XMAX, TICK, TMIN, TMAX, NTICK) — Determine first and last tick marks in a given range.

Subroutine Descriptions

GETELV (SKPLOT, NSMP, MOVAV1, MOVAV2, QNDPLT, HANDLE, IERR)
getelv.obj

Get elevation profiles from tape so they can be plotted. GETELV calls the subroutine RDTAPD to handle the peculiar file structure used on the tape, and performs the necessary second integration if the plot is detailed (rather than quick-n-dirty). It calls LOPASS to perform the moving average filtering.

→ SKPLOT	integer*4	number of samples to skip before plotting. This number should be calculated as X/DX.
→ NSMP	integer*4	number of samples to plot.
→ MOVAV1	integer*4	number of samples in moving average.
→ MOVAV2	integer*4	number of samples in 1/2 moving average.
→ QNDPLT	logical	switch for quick-n-dirty plotting.
→ HANDLE	integer*2	handle for file with processed profile.
← IERR	integer*2	error code. 0=cool.

$INCLUDE:'SETCOM'	variables in this common block describe the layout of the data in the file. These include the number of channels in each sector, the number of points, full-words, etc. It also looks at the TSTTYP variable to see if it is plotting a road test or a bounce test.
$INCLUDE:'BUFCOM'	the signals read from the file are put into the array PCBUFR.

GETLEN (X, XLL, XUL, UNITS, TITLE, PROMPT, IRET) *getlen.obj*

Prompt the user for some type of length measure or range. GETLEN is used to get plot scales, baselengths, and so forth. The menu provided the user will have XLL as the first option, XUL as the last, and will include X in the middle. The user will be given a list of values to select from the menu, and also the options to *cancel* and to select a *custom* value that is not explicitly included on the list.

↔ X	real*4	number that is updated by the subroutine.
→ XLL	real*4	lower limit of allowable values for X.
→ XUL	real*4	upper limit of allowable values for X.
→ UNITS	char*8	name of units used for X.
→ TITLE	char*32	heading for menu used to get X from user.
→ PROMPT	char*60	prompt to use for *custom* entry.
← RET	integer*2	return code. 0=ok, 1=cancel.

GRCURS (ISTART, IPLT, KCURS, NPTS, IMAX, NPTOT, NPMAX,IUPDT, XMIN, XMAX, XSTART, DX, YMIN, YMAX, ICH) *plotsubs.obj*

Wait for the user to hit a key, then interpret any cursor keys, and update plot parameters as necessary.

→ ISTART	integer*4	offset (in file) to 1st point in plot.

↔ IPLT	integer*2	number of active plot (1 or 2).
↔ KCURS	integer*4	offset to present cursor position.
↔ NPTS	integer*4	number of points on the screen.
→ IMAX	integer*2	number of plots on screen (1 or 2).
→ NPTOT	integer*4	number of points in data file.
→ NPMAX	integer*4	max number of points that can be plotted. (This is a function of the common block size.)
← IUPDT	integer*2	return cod. 0 = I and KCURS updated; 1 = changed limits for one plot; 2 = changed limits for 2 plots; 3=quit.
↔ XMIN	real*4	minimum x value.
↔ XMAX	real*4	maximum x value.
→ XSTART	real*4	value of x at start of file (i=0).
→ DX	real*4	sample interval.
↔ YMIN	real*4	array with min y values for each channel in file.
↔ YMAX	real*4	array with max y values for each channel in file.
→ ICH	integer*2	array with id no's of plotted channels.

LABEL (X, STRING, L) *plotsubs.obj*

Convert a real number into a string for Halo.

→ X	real*4	number to be converted.
← STRING	char*10	string representation of X, with beginning and ending \ characters for Halo.
← L	integer*2	number of characters in STRING. (Not counting beginning and ending \'s.)

LOGO *logo.obj*

This subroutine draws the logo for the profilometer using Halo subroutines. It then waits for the user to press a key to continue. If the key pressed is p or P, a hard copy is made. A text file named LOGO. contains the coordinates and words that are displayed on the screen. If this file is not present, an error will occur.

PLOT (MODE, IACTIV, NCHAN, NPTS, ICH, IIS, ITOT, DX, XMIN, XMAX, XSTART, KCURS, YMIN, YMAX, NAME, UNITS, XNAME, XUNITS, GAIN, OFF, IUPDT, ISTART, NPTOT, NPMAX, TITLE) *plot.obj*

Plot data contained in common array using Halo subroutines. A cursor is displayed along with printed values of the data at that point. The subroutine plots the data and then waits for keyboard inputs from the user. Keys that request cursor movement or changes in scaling of the y axis are handled within the subroutine. Keys that request a change in the scaling of the x axis or *End* cause the subroutine to return.

→	MODE	integer*2	data type: 0=integer*2; 1=real*4.
↔	IACTIV	integer*2	the active plot (1 or 2) with the cursor.
→	NCHAN	integer*2	number of channels to be plotted (1 or 2).
→	NPTS	integer*4	number of points (per channel) to plot.
→	ICH	integer*2	array with id nos. of the channel(s) being plotted.
→	IIS	integer*4	offset (in array) to first point to be plotted.
→	ITOT	integer*2	number of channels in buffer.
→	DX	real*4	sample interval (x axis gain).
↔	XMIN	real*4	minimum limit for x values.
↔	XMAX	real*4	maximum limit for x values.
→	XSTART	real*4	value of x at start of file (i=0).
↔	KCURS	integer*4	offset in file to cursor position (0=1st sample).
↔	YMIN	real*4	array of min y limits for all (ITOT) channels.
↔	YMAX	real*4	array of max y limits for all (ITOT) channels.
→	NAME	char*8	array of names of all channels.
→	UNITS	char*8	array of names of units of all of the channels.
→	XNAME	char*8	name of variable plotted on the x axis.
→	XUNITS	char*8	name of units for variable plotted on the x axis.
→	GAIN	real*4	array of channel gains used for integer*2 data.
→	OFF	real*4	array of offsets of channels used for integer*2 data.
↔	IUPDT	integer*2	0=don't redraw; 1=rescale y axis on active plot; 2=both plots, 3=quit. The only values on exit are 2 and 3.
↔	ISTART	integer*4	offset (in file) to first point in plot array.
→	NPTOT	integer*4	number of samples in file.
→	NPMAX	integer*4	max number of points that can be plotted.
→	FNAME	char*30	title for plots.

$INCLUDE:'BUFCOM' the data being plotted are in the 2-D array IBUF (for integer*2 data) or the 2-D array RDATA (for real*4 data.)

PLTELV (HANDLE, QNDPLT) *plotelv.obj*

This subroutine is used when the user selects either detailed or fast (quick-n-dirty) plots from the VIEW AND PROCESS DATA menu. It calls subroutines to select how many profiles will be plotted, which baselengths to use for the moving average filter, and what ranges to plot. It calls the Halo subroutines to switch from a text display to a graphics display. It then calls the subroutine GETELV to get the data from the open file, and then the subroutine PLOT to plot that data. Depending on the return code from PLOT, it will either switch back to the text display and return, or call GETELV using a different range and plot the new range.

→ HANDLE	integer*2	handle to data file.
→ QNDPLT	logical	.true. if its a quick and dirty plot.

$INCLUDE:'BUFCOM'	the signals are read from the file and put into the array PCBUFR.
$INCLUDE:'SETCOM'	the subroutine uses the variables in this common block that describe the layout of the data in the file. These include the number of channels in each sector, the number of points, full-words, etc. It also looks at the TSTTYP variable to see if it is plotting a road test or a bounce test.
$INCLUDE:'STATCOM'	the name of the open file is contained in common, and is shown on the plots.
$INCLUDE:'HANDLES'	contains some integer*4 variables used to read from the data file.

PLTRAW (HANDLE) *plotraw.obj*

This subroutine is used when the user selects PLOT RAW DATA from the VIEW AND PROCESS DATA menu. It calls subroutines to select which signals will be plotted and what ranges to plot. It calls the Halo subroutines to switch from a text display to a graphics display. It then calls the subroutine RDTAPE to get the data from the open file, and then the subroutine PLOT to plot that data. Depending on the return code from PLOT, it will either switch back to the text display and return, or call RDTAPE using a different range and plot the new range.

$INCLUDE:'BUFCOM'	the signals are read from the file and put into the array IBUF.
$INCLUDE:'SETCOM'	the subroutine uses the variables in this common block that describe the layout of the data in the file. These include the number of channels and the number of points. It looks at the TSTTYP variable to see if it is plotting a road test or a bounce test.
$INCLUDE:'STATCOM'	the name of the open file is contained in common, and is shown on the plots.
$INCLUDE:'HANDLES'	contains some integer*4 variables used to read from the data file.

PLTRUT (HANDLE) *plotrut.obj*

This subroutine is used when the user selects PLOT RUT & ROUGHNESS from the VIEW AND PROCESS DATA menu. It calls subroutines to select which channels will be plotted, which baselengths to use for the moving average smoothing, and what ranges to plot. It calls the Halo subroutines to switch from a text display to a graphics display. It calls the subroutine RDTAPD to get the data from the open file. If it is speed or rut data, the subroutine LOPASS is used to filter the data. For roughness data, PLTRUT does the processing. The subroutine PLOT is then called to plot the smoothed data. Depending on the return code from PLOT, it will either switch back to the text display and return, or plot data from a different range.

→ HANDLE	integer*2	handle to data file.

$INCLUDE:'BUFCOM'	the signals are read from the file and put into the array PCBUFR.
$INCLUDE:'SETCOM'	the subroutine uses the variables in this common block that describe the layout of the data in the file. These include the number of channels in each sector, the number of points, full-words, etc. It also looks at the TSTTYP variable to see if it is plotting a road test or a bounce test.
$INCLUDE:'STATCOM'	the name of the open file is contained in common, and is shown on the plots.

PLTSEL (NCHAN, NAME, UNITS, XNAME, XUNITS, DX, XMIN, XSTART, XRANGE, YRANGE, YMXRNG, NPTS, NPMAX, NPTOT, KCURS, ICH) *plotsel.obj*

This subroutine prompts the user for the selection of channels and plotting ranges. It is called by the PLTRAW, PLTELV, and PLTRUT subroutines.

→ NCHAN	integer*2	number of channels.
→ NAME	char*8	array with names of each channel.
→ UNITS	char*8	array with units for each channel.
→ XNAME	char*8	name of variable plotted on x axis (time, etc.).
→ XUNITS	char*8	name of units for x axis.
→ DX	real*4	sample interval.
↔ XMIN	real*4	minimum limit of plotting range.
→ XSTART	real*4	x value at start of file (i=0).
↔ XRANGE	real*4	plotting range for x axis.
↔ YRANGE	real*4	array with plotting ranges for y axis.
→ YMXRNG	real*4	array with max allowable range for each channel.
← NPTS	integer*4	number of points to plot.
→ NPMAX	integer*4	maximum number of points that can be plotted.
→ NPTOT	integer*4	maximum number of points in file.
↔ KCURS	integer*4	position of cursor in file (0=1st point).
↔ ICH	integer*2	array containing the 2 channels to be plotted.

SCLDWN (X, XNORM, XDOWN) *plotsubs.obj*

This subroutine scales a variable down so that it has only one significant digit, with that digit being a 1, 2, or 5. (For example, 23.4 would be scaled down to 20; 0.07 would be scaled down to 0.05.)

→ X	real*4	number to be scaled up.
← XNORM	real*4	normalized value for X.
← XDOWN	real*4	scaled down value for X.

SCLUP (X, XNORM, XUP) *plotsubs.obj*

This subroutine scales a variable up so that it has only one significant digit, with that digit being a 1, 2, or 5. (For example, 23.4 would be scaled up to 50; 0.07 would be scaled up to 0.1.)

→ X	real*4	number to be scaled up.
← XNORM	real*4	normalized value for X.
← XUP	real*4	scaled up value for X.

TIKSET (XMIN, XMAX, TICK, TMIN, TMAX, NTICK) *plotsubs.obj*

Determine first and last tick marks in a given range.

→ XMIN	real*4	minimum limit in range (eng. units).
→ XMAX	real*4	maximum limit to range (eng. units).
→ TICK	real*4	tick interval (eng. units).
← TMIN	real*4	first tick interval within range (eng. units).
← TMAX	real*4	last tick interval within range (eng. units).
← NTICK	integer*2	number of ticks within range.

Printing

These subroutines support the menu option to PRINT NUMERICS.

Quick Reference

PRTLF (LSCR, LLPT, LFL) — Add carriage returns after each line.
PRTNUM (HANDLE) — Print numerics averaged over a specified interval.
PUTYN (YESNO, IROW, ICOL) — Put Y or N in specified screen location.
YESNOL (YESNO,IROW,ICOL,IRET) — Get Yes/No answer and set logical variable.

PRTLF (LSCR, LLPT, LFL) *prtnum.obj*

Add carriage returns after each line.

→ LSCR	logical	.true. if PRTNUM is currently printing to the screen.
→ LLPT	logical	.true. if PRTNUM is currently printing to the printer (Fortran IO unit #6).
→ LFL	logical	.true. if PRTNUM is currently printing to a file (Fortran IO unit #7).

PRTNUM (HANDLE) *prtnum.obj*

This subroutine allows the user to print numerics averaged over a specified interval, based on the channels stored in the rut part of a data file. (These channels are the rut depth signals, the IRI roughness signals, and the decimated velocity channel.) The printouts can be shown on the screen, sent to the printer, and sent to a text file. This subroutine requires a file called PRTSCR that contains the text and coordinates used for the screen that shows the options to the user.

→ HANDLE	integer*2	handle for data file.

$INCLUDE:'SETCOM'	the subroutine uses the variables in this common block that describe the layout of the processed data in the file. These include the number of channels in each sector, the number of points, 4-byte reals, etc.
$INCLUDE:'BUFCOM'	the signals are read from the file and put into the array PCBUFR, where they are averaged as needed for printing.
$INCLUDE:'STATCOM'	contains the name of the open file.

PUTYN (YESNO, IROW, ICOL) *prtnum.obj*

Put Y or N in specified screen location, based on logical variable YESNO.

→ YESNO	logical	if .true., put Y. If .false., put N.
→ IROW	integer*2	row on text screen to locate the Y/N character.
→ ICOL	integer*2	column on text screen to locate the Y/N character.

YESNOL (YESNO,IROW,ICOL,IRET) *prtnum.obj*

Get Yes/No answer and set logical variable. (Similar to the YESNO subroutine in the Fortran Extensions described in appendix C.)

↔ YESNO	logical	variable that is updated by the subroutine.

→ IROW	integer*2	row on text screen to locate the Y/N character.
→ ICOL	integer*2	column on text screen to locate the Y/N character.
← IRET	integer*2	return code that tells which cursor key was used to accept YESNO value. (Same values as used in Fortran extension subroutines.)

Program Control

This subsection describes the main profile program and the major subroutines that create the menus used to select the various options available for the profilometer.

Quick Reference

BATCH (DR) — Process a list of data files.
GOAHED (HANDLE) — Warn the user that some processing needs to be done.
IOEX — Present a menu of options to exercise the input/output hardware.
MAIN — Show the Logo, then offer the main menu to the user.
MEASURE — Generate the menu for measuring data and call the appropriate subroutines.
PROCESS — Generate the menu for viewing data and call the appropriate subroutines.

Subroutine Descriptions

BATCH (DR) *batch.obj*

Process a list of data files. The signal processing subroutines CHKSAT and PRFCMP are used as needed to convert files with raw data to files with profile and rut depth.

↔ DR	char*1	letter indicating current drive.

$INCLUDE:'HANDLES'	contains some integer*4 variables used to read from the data file.
$INCLUDE:'SETCOM'	includes the variables TSTTYP and ITSOK, used to enable and disable menu options based on the status of the open file.

GOAHED (HANDLE) *process.obj*

Warn the user that some processing needs to be done and that it might take a few minutes. If the user answers yes, process the data.

→ HANDLE	integer*2	handle of open file.

$INCLUDE:'SETCOM'	the subroutine looks at the variables TSTTYP and ITSOK to see what needs to be done with the file.

IOEX *ioex.obj*

Presents a menu of options to exercise the input/output hardware of the profilometer. These options are used in tracing hardware problems and performing comprehensive calibrations.

$INCLUDE:'IOPARMS' defines addresses or constants related to the hardware.

MAIN *profmain.obj*

The main program first shows the logo, and after a key has been pressed, performs some once-only initializations. Then it offers the main menu to the user.

$INCLUDE:'BUFCOM'
$INCLUDE:'SETCOM'
$INCLUDE:'STATCOM' includes the date, time since last calibration, current file name, and other status information in a common block.

MEASURE *measure.obj*

Generates the menu for measuring data and calls the appropriate subroutines based on the items selected from that menu.

$INCLUDE:'STATCOM' includes the date, time since last calibration, current file name, and other status information in a common block.

PROCESS *process.obj*

Generate the menu for viewing data and call the appropriate subroutines based on the items selected from that menu. It first writes all of the current status information (from the STATCOM common block) into a temporary file, which is restored before exiting the subroutine.

$INCLUDE:'STATCOM' includes the current file name.
$INCLUDE:'HANDLES' contains some integer*4 variables used to read from the data file.
$INCLUDE:'SETCOM' the subroutine mainly uses the variables TSTTYP and ITSOK to enable and disable menu options based on the status of the open file.

Signal Processing

This subsection describes the subroutines used to process the measured transducer signals and the other signals derived from them.

Quick Reference

AVEVEL (IBUF, NC1, NS, RBUF, NC2, TRIM, GAIN, BIAS) — Average and decimate a (speed) signal.
CHKSAT (HANDLE, AUTO) — Check the raw transducer signals for saturation.
DEBIAS (ARRAY, NCH, NS, BIAS) — subtract bias from signal in real*4 array.
HIPASS (ARRAY, NCH, N1, N2, N3, N4, N5, MOVAV1, MOVAV2) — Filter a signal with a high-pass filter.
Function IAVE (ARRAY, NCH, NS) — Average value of signal in integer*2 array.
LOPASS (ARRAY, NCH, NS, MOVAV1, MOVAV2) — Smooth a signal.
LRSLOP (ARRAY, NDIM, NSAMP, SLOPE) — Calculate slope of signal using a linear regression.
MINV (ARRAY, N, D, LARRAY, MARRAY) — Matrix inversion.
PRFCMP (HANDLE) — Convert raw data into slope profile, rut depth, IRI roughness, and elevation profile.
PRFELV (BUF1, NC1, NS, BUF2, NC2, TRIM, DX, C, ENDELV) — Compute compressed elevation profile from slope.
PRFIRI (BUF1, BUF2, X1, X2, X3, X4, ROUGH) — Filter a slope profile signal using the IRI quarter-car simulation.
Function RAVE (ARRAY, NCH, NS) — Average value of signal in real*4 array.
RUTCMP (HL, HC, HR, NCHRAW, NS, RUT, NCHRUT, TRIM, GAINL, GAINC, GAINR, ZL, ZC, ZR, HLLAT, HRLAT) — Compute, average, and decimate a rut depth signal.
SATMAX (ARRAY, NCH, NS, OFFSET, MAX, COUNT, NSAT, LSAT) — Check raw data signal for saturation at upper limit.
SATMIN (ARRAY, NCH, NS, OFFSET, MIN, COUNT, NSAT, LSAT) — Check raw data signal for saturation at lower limit.
SETSTM — Calculate coefficients for quarter-car simulation.

Subroutine Descriptions

AVEVEL (IBUF, NC1, NS, RBUF, NC2, TRIM, GAIN, BIAS) *sigsubs.obj*

Average and decimate a (speed) signal.

→	IBUF	integer*2	2-D input array. Channel 1 is processed.
→	NC1	integer*4	1st dimension (# of channels) for IBUF.
→	NS	integer*4	2nd dimension (# of samples) for IBUF.
←	RBUF	real*4	2-D output array. Channel 1 is processed.

→ NC2	integer*4	1st dimension (# of channels) for RBUF.
→ TRIM	integer*4	decimation ratio. Every TRIM-th point is kept.
→ GAIN	real*4	gain for input data: engineering units/count.
→ BIAS	real*4	bias in input data.

CHKSAT (HANDLE, AUTO) *chksat.obj*

Check the raw transducer signals for saturation.

→ HANDLE	integer*2	handle for data file that gets checked.
→ AUTO	integer*2	code indicating interactive or auto modes. 0 = interactive, 1 = don't truncate if error, 2 = truncate if error, 3 = interactive if error.

$INCLUDE:'BUFCOM'	the signals are read from the file and put into the array PCBUFI for checking.
$INCLUDE:'HANDLES'	contains some integer*4 variables used to read from the data file.
$INCLUDE:'SETCOM'	the subroutine uses the variables in this common block that describe the layout of the data in the file. These include the number of channels and the number of points. It updates the variables TSTTYP, VELMIN, VELMAX, ITSOK, and the arrays NSAT and LSAT to indicate the results of the check.

DEBIAS (ARRAY, NCH, NS, BIAS) *sigsubs.obj*

Subtract bias from signal in real*4 array.

↔ ARRAY	real*4	2-D array. Channel 1 is processed.
→ NCH	integer*4	1st dimension (# of channels) for ARRAY.
→ NS	integer*4	2nd dimension (# of samples) for ARRAY.
→ BIAS	real*4	bias to be subtracted from channel #1 in ARRAY.

HIPASS (ARRAY, NCH, N1, N2, N3, N4, N5, MOVAV1, MOVAV2) *sigsubs.obj*

This subroutine filters a signal with a hipass filter based on the moving average. If necessary, the subroutine adds points to the beginning and end of the signal so that the moving average filter can be applied over the entire length of the signal.

↔ ARRAY	real*4	2-D Input array. Channel 1 is filtered. This array must be dimensioned to cover (N1 + (N2 + N3 + N4 + N5 + MOVAV1 + 1) samples. The input data should start at the second position and continue to the end of the array. The

		output starts at the first position, and continues to the N3-th position.
→ NCH	integer*4	1st dimension of ARRAY (# of channels).
→ N1-N5	integer*4	no. of samples in five contiguous regions of memory.
→ MOVAV1	integer*4	no. of points in moving average.
→ MOVAV2	integer*4	no. of points to center of moving average (MOVAV1 / 2).

Function IAVE (ARRAY, NCH, NS) *sigsubs.obj*

Average value of signal in integer*2 array.

← IAVE	integer*2	average value of channel-1 in ARRAY.
→ ARRAY	integer*2	2-D input array. Channel 1 is processed.
→ NCH	integer*4	1st dimension (# of channels) for ARRAY.
→ NS	integer*4	2nd dimension (# of samples) for ARRAY.

LOPASS (ARRAY, NCH, NS, MOVAV1, MOVAV2) *lopass.obj*

This subroutine filters a signal using a moving average as a smoothing (lopass) filter.

↔ ARRAY	real*4	2-D Input array. Channel 1 is filtered. The data should start at position 2 and continue to NS + 1. The output starts in position 1, and corresponds to what used to be the MOVAV1-th point. (The array gets shifted.)
→ NCH	integer*4	1st dimension of ARRAY (# of channels).
→ NS	integer*4	no. of samples in ARRAY.
→ MOVAV1	integer*4	no. of points in moving average.
→ MOVAV2	integer*4	no. of points to center of moving average (MOVAV1 / 2).

LRSLOP (ARRAY, NDIM, NSAMP, SLOPE) *sigsubs.obj*

Calculate slope of signal using a linear regression.

→ ARRAY	real*4	2-D Input array.
→ NDIM	integer*4	1st dimension of ARRAY. (# of channels.)
→ NSAMP	integer*4	2nd dimension of ARRAY. (# of samples.)
← SLOPE	real*4	slope of channel 1 in ARRAY as obtained by linear regression against the sample number.

MINV (ARRAY, N, D, LARRAY, MARRAY) *minv.obj*

Matrix inversion subroutine, taken from the IBM SSP library. The code has been modified as necessary to compile under Fortran 77.

↔ ARRAY	real*4	general 2-D square (N x N) array that is inverted.
→ N	integer*2	order of the ARRAY matrix.
← D	real*4	determinant of ARRAY.
← LARRAY	integer*2	1-D array needed by MINV, dimensioned to length N.
← MARRAY	integer*2	1-D array needed by MINV, dimensioned to length N.

PRFCMP (HANDLE) *prfcmp.obj*

This subroutine does the basic signal processing, converting a data file from an integer*2 representation of raw data into 3 interleaved real*4 representations of slope profile, rut depth and IRI roughness, and elevation profile. The subroutine uses the RDTAPE and WRTAPE subroutines to access the file. Progress of the processing is displayed on the screen.

→ HANDLE	integer*2	handle for data file that gets processed.

$INCLUDE:'SETCOM'	the subroutine uses the variables in this common block that describe the raw data, and sets the parameters describing the layout of the processed data in the file. These include the number of channels in each sector, the number of points, full-words, etc.
$INCLUDE:'BUFCOM'	the signals are read from the file and initially put into the array PCBUFI for checking. The array PCBUFR is used for the computed slope profiles and other output signals.

PRFELV (BUF1, NC1, NS, BUF2, NC2, TRIM, DX, C, ENDELV) *sigsubs.obj*

Subroutine to compute compressed elevation profile from slope.

→ BUF1	real*4	2-D input array. Channel 1 is processed.
→ NC1	integer*4	1st dimension (# of channels) for BUF1.
→ NS	integer*4	2nd dimension (# of samples) for BUF1.
← BUF2	real*4	2-D output array. Channel 1 is processed.
→ NC2	integer*4	1st dimension (# of channels) for BUF2.
→ TRIM	integer*4	decimation ratio. Every TRIM-th point is kept.
→ DX	real*4	sample interval for BUF1.
→ C	real*4	coefficient to add high-pass filtering to the integration.
↔ ENDELV	real*4	as input, starting elevation at beginning of buffer. as output, elevation at end of buffer.

PRFIRI (BUF1, BUF2, X1, X2, X3, X4, ROUGH) *prfiri.obj*

This subroutine filters a slope profile signal using the IRI quarter-car simulation. The accumulated IRI roughness is compressed and stored in a separate array. The IRI coefficients and the sizes of the arrays are obtained from COMMON. This subroutine will probably be enhanced to smooth the slope profiles, so it should not be called until the profiles are stored on tape.

→ BUF1	real*4	2-D input array with profile data. Ch-1 is processed.
← BUF2	real*4	2-D output array (with rut stuff also.) Ch-1 is replaced.
↔ X1-X4	real*4	vehicle response variables, updated every step.
↔ ROUGH	real*4	accumulated roughness, updated every step.

$INCLUDE:'SETCOM'	the subroutine uses the variables in this common block that describe the layout of the processed data in the file. These include the number of channels in each sector, the number of points, full-words, etc.

Function RAVE (ARRAY, NCH, NS) *sigsubs.obj*

Average value of signal in real*4 array.

← **RAVE**	real*4	average value of channel #1 in ARRAY.
→ ARRAY	real*4	2-D input array. Channel 1 is processed.
→ NCH	integer*4	1st dimension (# of channels) for ARRAY.
→ NS	integer*4	2nd dimension (# of samples) for ARRAY.

RUTCMP (HL, HC, HR, NCHRAW, NS, RUT, NCHRUT, TRIM, GAINL, GAINC, GAINR, ZL, ZC, ZR, HLLAT, HRLAT)

rutcmp.obj

Compute, average, and decimate a rut depth signal.

→ HL	integer*4	2-D array with left-hand height signal.
→ HC	integer*2	2-D array with center (in rut) height signal.
→ HR	integer*2	2-D array with right-hand height signal.
→ NCHRAW	integer*2	number of raw data channels. (HL, HC, HR are channels in the same 2-D array.)
→ NS	integer*4	number of samples (before decimation).
← RUT	real*4	2-D array for output rut depth signal.
→ NCHRUT	integer*4	number of channels in output array.
→ TRIM	integer*4	decimation ratio.
→ GAINL	real*4	gain for left-hand height signal.
→ GAINC	real*4	gain for center height signal.
→ GAINL	real*4	gain for right-hand height signal.

→ ZL	real*4	height of L. height sensor when it reads zero.
→ ZC	real*4	height of C. height sensor when it reads zero.
→ ZR	real*4	height of R. height sensor when it reads zero.
→ HLLAT	real*4	lateral distance between L. and C. sensors.
→ HRLAT	real*4	lateral distance between R. and C. sensors.

SATMAX (ARRAY, NCH, NS, OFFSET, MAX, COUNT, NSAT, LSAT) *chksat.obj*

Check a raw data signal for saturation at an upper limit. To do this, find the maximum value and also look for two or more consecutive samples at that limit.

→ ARRAY	integer*2	2-D input array. Channel 1 is checked.
→ NCH	integer*4	number of channels in ARRAY.
→ NS	integer*4	number of samples in ARRAY.
→ OFFSET	integer*4	number of samples previously processed.
↔ MAX	integer*2	maximum value in signal. The subroutine will update this value if a larger amplitude is found.
↔ COUNT	integer*4	counter used to see if signal stays at max level for two adjacent samples.
↔ NSAT	integer*4	number of saturations in signal.
↔ LSAT	integer*4	location (sample no.) of first saturation.

SATMIN (ARRAY, NCH, NS, OFFSET, MAX, COUNT, NSAT, LSAT) *chksat.obj*

Check a raw data signal for saturation at a lower limit. To do this, find the maximum value and also look for two or more consecutive samples at that limit.

→ ARRAY	integer*2	2-D input array. Channel 1 is checked.
→ NCH	integer*4	number of channels in ARRAY.
→ NS	integer*4	number of samples in ARRAY.
→ OFFSET	integer*4	number of samples previously processed.
↔ MIN	integer*2	minimum value in signal. The subroutine will update this value if a smaller amplitude is found.
↔ COUNT	integer*4	counter used to see if signal stays at min level for two adjacent samples.
↔ NSAT	integer*4	number of saturations in signal.
↔ LSAT	integer*4	location (sample no.) of first saturation.

SETSTM *setstm.for*

This subroutine calculates coefficients for the state-transition matrix used in the IRI quarter-car simulation. It requires MINV, a matrix inversion subroutine.

$INCLUDE:'SETCOM' the SETUP array includes the sample interval and the arrays containing the quarter-car coefficients.

Test and Calibration

This subsection describes the subroutines that collect data for calibration and testing applications.

Quick Reference

ACAL (ICHAN, ROW)—Calibrate an analog data channel.
ADSET (ADCURB, BUFT, BUFST, NBUF, BYTB, MAXB, BUFFCNT, DONE)—Set up the data collection parameters and the interrupt routine.
CALIB—Calibrate the analog hardware and check the height sensors.
CONFIGURE—Select which data to collect.
PULSE—Check the calibration of the distance sender.
PULTST (PASS, DONE, JJ, CONV, MAXP)—Set up the interrupt and data collection routine for the distance pulser check.
STARTAD (FF, BUFST, BUFT, BUFFCNT, MAXB, ADCURB, DONE, INDEX)—Start the data collection.
TCHECK (IC, ROW, IPOS)—Check a height transducer.
TEST (IITY)—Collect data.

Subroutine Descriptions

ACAL (ICHAN, ROW) *calib.obj*

Calibrate one analog channel. In this routine, all readings are actually averages of one second of data sampled at 300 hertz. First, the current zero data voltage is read and along with the RMS noise, printed on the screen. The offset D/A on the analog card is then loaded so that the zero data voltage is within 40 millivolts of zero. The adjusted value and its associated noise are also printed on the screen. Next, the transducer is disconnected and a calibration D/A signal is put into the amplifier. A staircase signal is put in and the amplifier outputs are simultaneously measured. The actual amplifier gains are calculated from a regression of this data. The gains and resulting full-scale values are the written to the screen.

→ ICHAN	integer*2	channel number.
→ ROW	integer*2	row on the screen where the calibration results are printed.

INCLUDE:'SETCOM' contains the analog channel names, zero data values, and gains measured by the calibration.

ADSET (ADCURB, BUFT, BUFST, NBUF, BYTB, MAXB, BUFFCNT, DONE)

adnew.obj

This subroutine is written in assembly language. It saves the addresses of the data acquisition parameters for the interrupt routine (also included in adnew.obj), sets up the interrupt vector via a DOS call, and enables the interrupt on the 8259 interrupt controller. Data are collected automatically via DMA and the A/D clock sequencer.

←	ADCURB	integer*2	current A/D buffer being filled.
→	BUFT	integer*4	array containing the buffer addresses.
↔	BUFST	integer*2	array containing the status of the buffers. 0= buffer empty, −1 = buffer is full.
→	NBUF	integer*2	number of buffers.
→	BYTB	integer*2	number of bytes per buffer.
→	MAXB	integer*2	maximum number of buffers to fill.
→	BUFFCNT	integer*2	number of buffers filled.
↔	DONE	integer*2	status flag. 0= not done, -1=done, 1=error

CALIB

calib.obj

This subroutine calibrates all the channels previously selected by the configure subroutine. With the calibration bar in the middle or zero data position, it does an electrical calibration via a call to ACAL. With the calibration bar in the top and bottom positions, it subsequently calls TCHECK to check the accuracy of the height transducers. It then updates all of the calibration status variables

INCLUDE:'SETCOM'	contains the analog channel names, zero data values, and gains measured by the calibration.
INCLUDE:'STATCOM'	contains the calibration status variables.

CONFIGURE

config.obj

Displays the configuration menu to the operator and allows the selection of which data channels are to be collected (e.g., "left profile" or "left profile and left rut"). The subroutine reads in the number of channels, A/D start and stop channels, buffer offsets, and other variables used in processing the data from the file CONFIG.SET.

INCLUDE:'SETCOM'	the set array contains the configuration information.

PULSE

pulse.obj

This subroutine checks the accuracy of the wheel pulser by comparing a known distance traveled with the measured distance provided by the pulser. The average velocity derived from the distance pulser is also compared with the known velocity (known distance traveled divided by the elapsed time). The pulser distance, true distance, measured velocity, and true velocity are printed on the screen.

INCLUDE:'BUFCOM'	data are collected into the array IBUF.
INCLUDE:'IOPARMS'	i/o parameters are required to enable data collection.
INCLUDE:'SETCOM'	contains the gain for the pulser and the gain and offset for the velocity channel.

PULTST (PASS, DONE, JJ, CONV, MAXP) *pulsetst.obj*

This subroutine is written in assembly language. It saves the addresses of the data acquisition parameters for the interrupt routine (also included in adnew.obj), sets up the interrupt vector via a DOS call, and enables the interrupt on the 8259 interrupt controller. Data are collected both via DMA and by programmed I/O without the A/D sequencer.

↔ PASS	integer*4	pass count.
↔ DONE	integer*2	flag. 0=not done, -1=done 1=error.
→ PHSAD	integer*4	physical buffer address.
→ CONV	integer*2	number of channels or conversion per pass.
→ MAXP	integer*4	maximum number of passes.

STARTAD (FF, BUFST, BUFT, BUFFCNT, MAXB, ADCURB, DONE, INDEX) *startad.obj*

This subroutine is called by TEST to start the data collection process. First, it calculates the beginning of the buffers so that there won't be any page overruns. Then, it stores the buffer addresses in the array BUFT and initializes the status array BUFST. It resets the A/D sequencer and then sets it up for the current configuration. Finally, it sets the filter clock, call ADSET to set up the interrupt routine, and then waits for a key to be pressed. When a key is pressed data collection begins and the subroutine returns.

→ FF	real*4	filter clock frequency.
↔ BUFST	integer*2	array containing the status of the buffers. 0= buffer empty, −1 = buffer is full.
↔ BUFT	integer*4	array containing the buffer addresses.
→ BUFFCNT	integer*2	number of buffers filled.
→ MAXB	integer*2	maximum number of buffers to fill.
← ADCURB	integer*2	current A/D buffer being filled.
↔ DONE	integer*2	status flag. 0= not done, -1=done, 1=error.
← INDEX	integer*2	index into the array IBUF where the buffers start.

INCLUDE:'BUFCOM'	data are collected into the array IBUF.
INCLUDE:'IOPARMS'	I/O parameters are required to enable data collection.
INCLUDE:'SETCOM'	contains number of channels and other data collection parameters.

TCHECK (IC, ROW, IPOS) *calib.for*

This subroutine writes the nominal height of a transducer to the screen and then measures the actual height, prints it, and compares the two and prints a warning if they don't agree within two percent.

→ IC	integer*2	transducer channel number.
→ ROW	integer*2	row on the screen where results are printed.
→ IPOS	integer*2	position of the bar 1=top 2=bottom.

INCLUDE:'SETCOM' contains the gains and offsets of the height transducers.

TEST (IITY) *test.obj*

This subroutine controls the collection of road data. First the operator can edit the test display screen. Information such as surface type, lane, direction traveled, route, test speed, sample length, etc,. can be changed. The routine checks for an initialized tape and ensures that the given test will fit on the current drive of the tape. The test begins when the operator hits any key. Data are collected into fifteen buffers and written to tape a buffer at a time during data collection. After a buffer is written it can be filled again. The test ends when: (1) the operator hits a key, (2) the test length has been reached, or (3) there is no buffer available to write into. The remaining data are then recorded and TEST returns. If it is a normal test, data collection is triggered by the wheel pulser and A/D sequencer. If it is a bounce test, data are sampled at a fixed time interval.

→ IITY	integer*2	test type 0=normal test 1=bounce test.

INCLUDE:'BUFCOM'	data are collected into the array IBUF.
INCLUDE:'HANDLES'	contains definition of variables for file creating, positioning and writing.
INCLUDE:'IOPARMS'	I/O parameters are required to enable and disable data collection.
INCLUDE:'SETCOM'	contains number of channels and other data collection parameters.
INCLUDE:'STATCOM'	contains tape status variables.

APPENDIX A: SCHEMATICS

This appendix contains schematic drawings for the hardware fabricated at UMTRI. Some of the cards had been designed prior to the profilometer, and are used for other applications. In these cases, the drawings may show optional components that were not required for the configurations appropriate to a profilometer. For example, the schematic for the analog signal-conditioning card in figure 23 shows several optional trim pots that were not installed for the profilometer.

Table 10. Wiring list for the signal-conditioning unit backplane bus wiring.

Pin #	Control Card	Cards C0 - C15	Pull up Card	Function
A+1				+5V
B+2				5V Gnd
C 3				SCD
D 4				CAL. HIGH CAL. LOW
E 5				
F 6				
H 7				
J 8				
K 9				
L 10				
M 11				
N+12				-15V
P+13				COM
R+14				+15V
S 15				D7 D6
T 16				D5 D4
U 17				D3 D2
V 18				D1 D0
W 19				DHS
X 20				+REF FILTER CLK
Y 21				
Z 22				

DE9-S ChO	Signal when used with an Analog Signal Conditioning Card	22/44 Edge Connector CO
5	-15V	11
4	+15V	M
9	COM.	10-L
8	EXC.	9-K
2	-SENSE	8
1	+SENSE	J
6	-SIGNAL	7
7	+SIGNAL	H
3	COM.	13

Note: This is for channel 0, channels 1-15 are the same.

Figure 22. Schematic of the signal-conditioning unit backplane transducer wiring.

Table 11. Wiring list for the signal-conditioning unit backplane control wiring.

From Control Card Pin#	to Card-Pin#	From Control Card Pin#	to Card-Pin#	From DB25 Connector	to
3	C0-C	15	C0-19	A-1	DIP-1
4	C1-C	16	C1-19	A-2	DIP-2
5	C2-C	17	C2-19	A-3	DIP-3
6	C3-C	18	C3-19	A-4	DIP-4
7	C4-C	19	C4-19	A-5	DIP-5
8	C5-C	20	C5-19	A-6	DIP-6
9	C6-C	21	C6-19	A-7	DIP-7
10	C7-C	22	C7-19	A-8	DIP-8
C	C8-C	S	C8-19	A-11	C0-3
D	C9-C	T	C9-19	A-12	C0-4
E	C10-C	U	C10-19	A-13	C0-D
F	C11-C	V	C11-19	B-13	C0-S
H	C12-C	W	C12-19	B-12	C0-15
J	C13-C	X	C13-19	B-11	C0-T
K	C14-C	Y	C14-19	B-10	C0-16
L	C15-C	Z	C15-19	B-9	C0-U
				B-8	C0-17
				B-7	C0-V
DB25/A 14-25 to GND				B-6	C0-18
DB25/B 14-25 to GND				B-5	C0-W
DIP 9-16 to GND				B-4	C0-20
				B-3	C0-Y
				B-2	C0-21

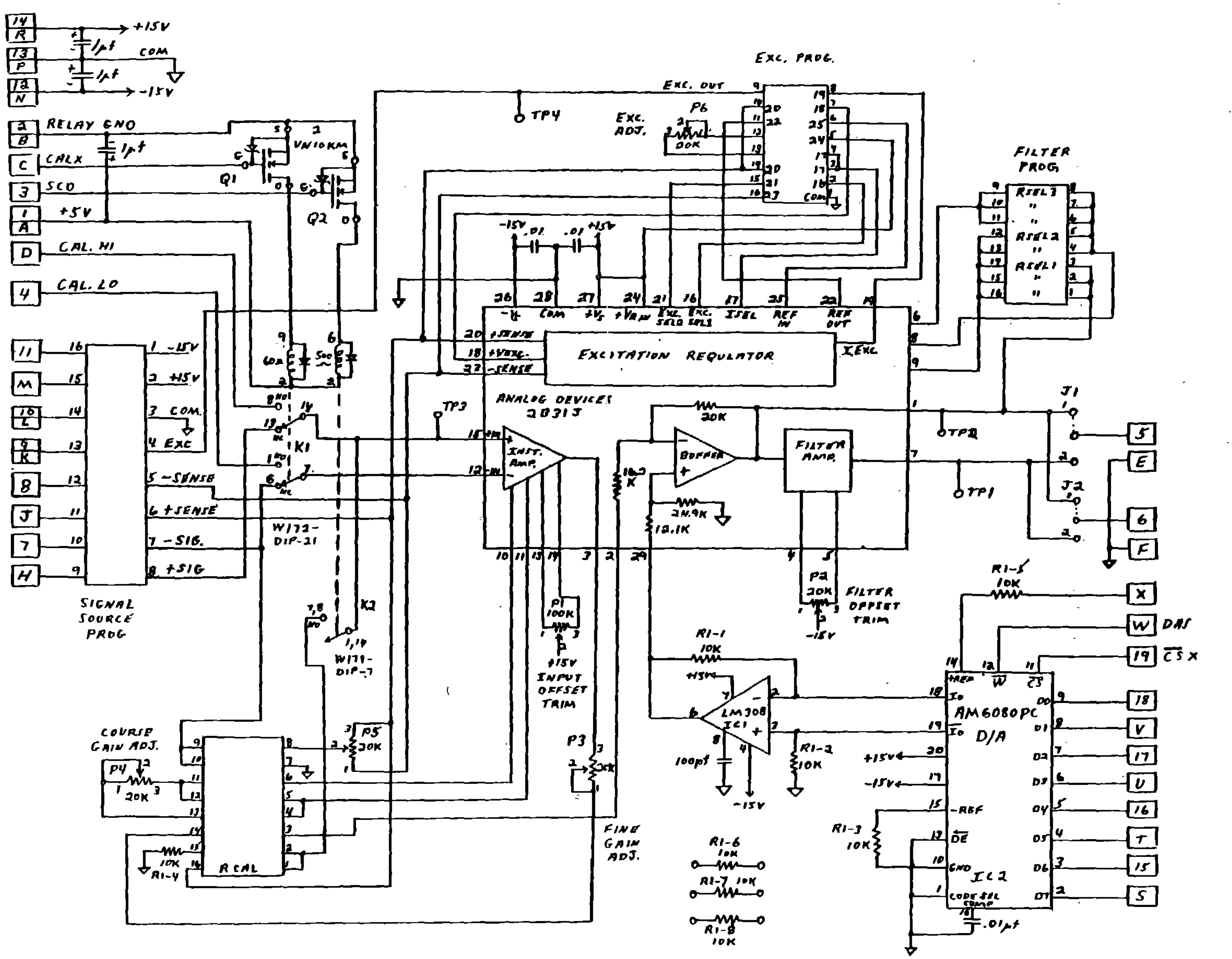


Figure 23. Schematic of an analog signal-conditioning card.

Table 12. Headers for the analog signal-conditioning cards.

CH	GAIN	I/O	EXCITATION	FILTER
0 HGT RGHT & 4 HGT LEFT & 5 MID RUT	3-14 Short 5-12 Short 6-11; 91K; G=2.03 8-9 Short	8-9 Short 3-14 Short	1-16 Short 2-15 Short 6-13 Short 7-9&10 Short 11-12 Short	None
1 AZ RGHT & 3 AZ LEFT	3-14 Short 5-12 Short 6-11; 91K; G=2.03 8-9 Short	8-9 Short 3-14 Short 2-15 Short 1-16 Short 8-3; 2K 1%	1-16 Short 2-15 Short 6-13 Short 7-9&10 Short 11-12 Short	None
2 VELOCITY	3-14 Short 5-12 Short 6-11; 91K; G=2.03 7-10 Short	8-9 Short	None	None

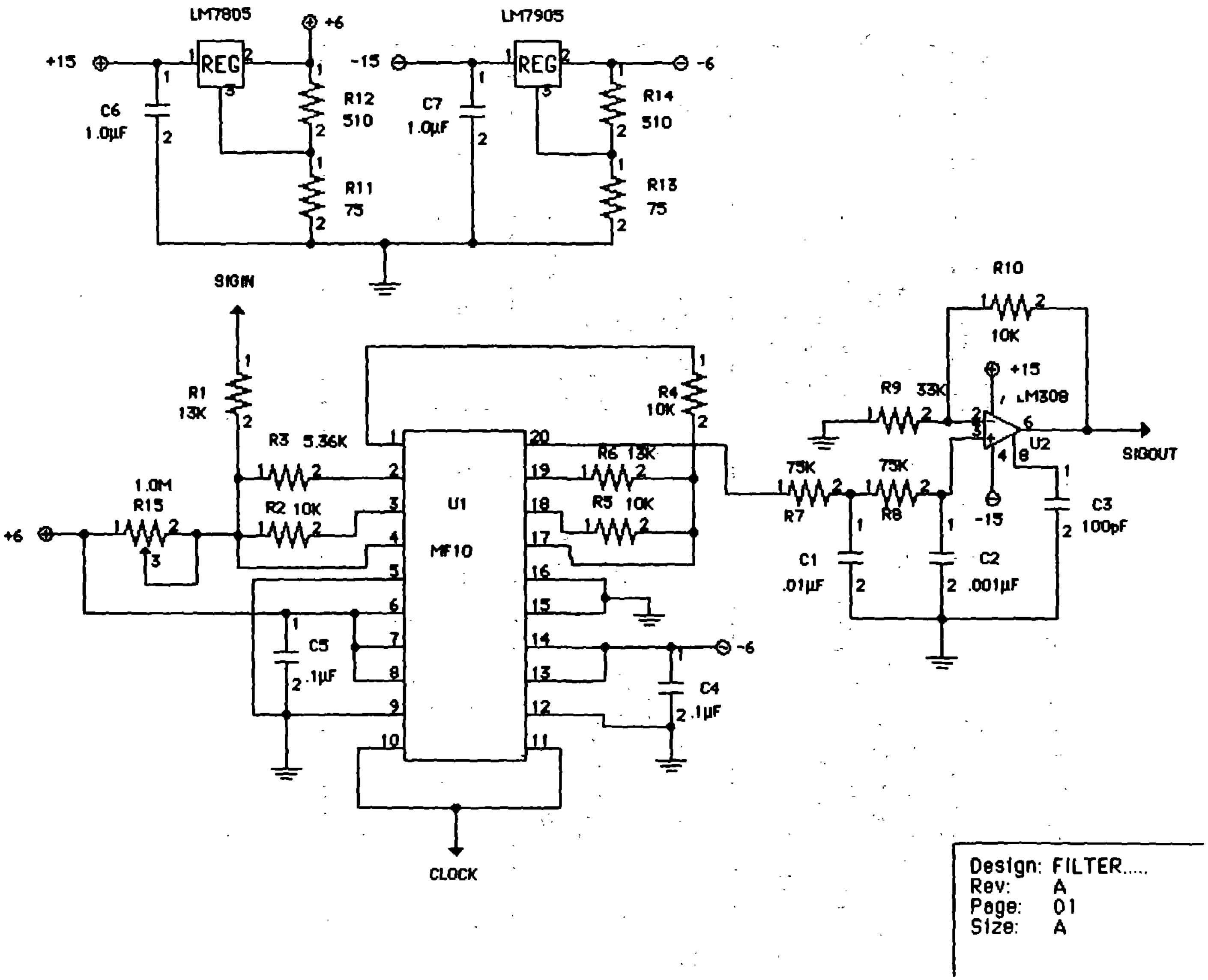


Figure 24. Schematic of a filter card.

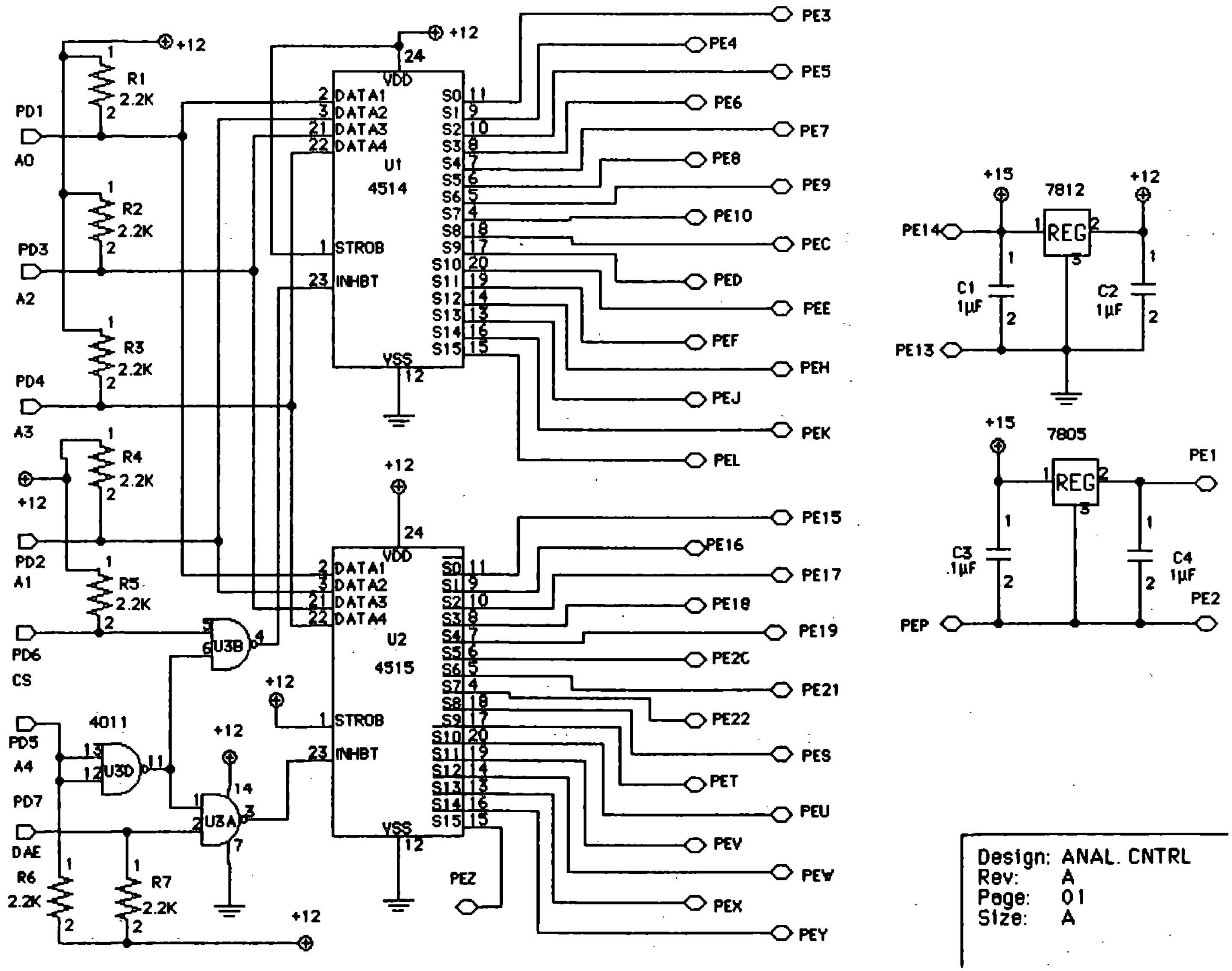


Figure 25. Schematic of the analog control card.

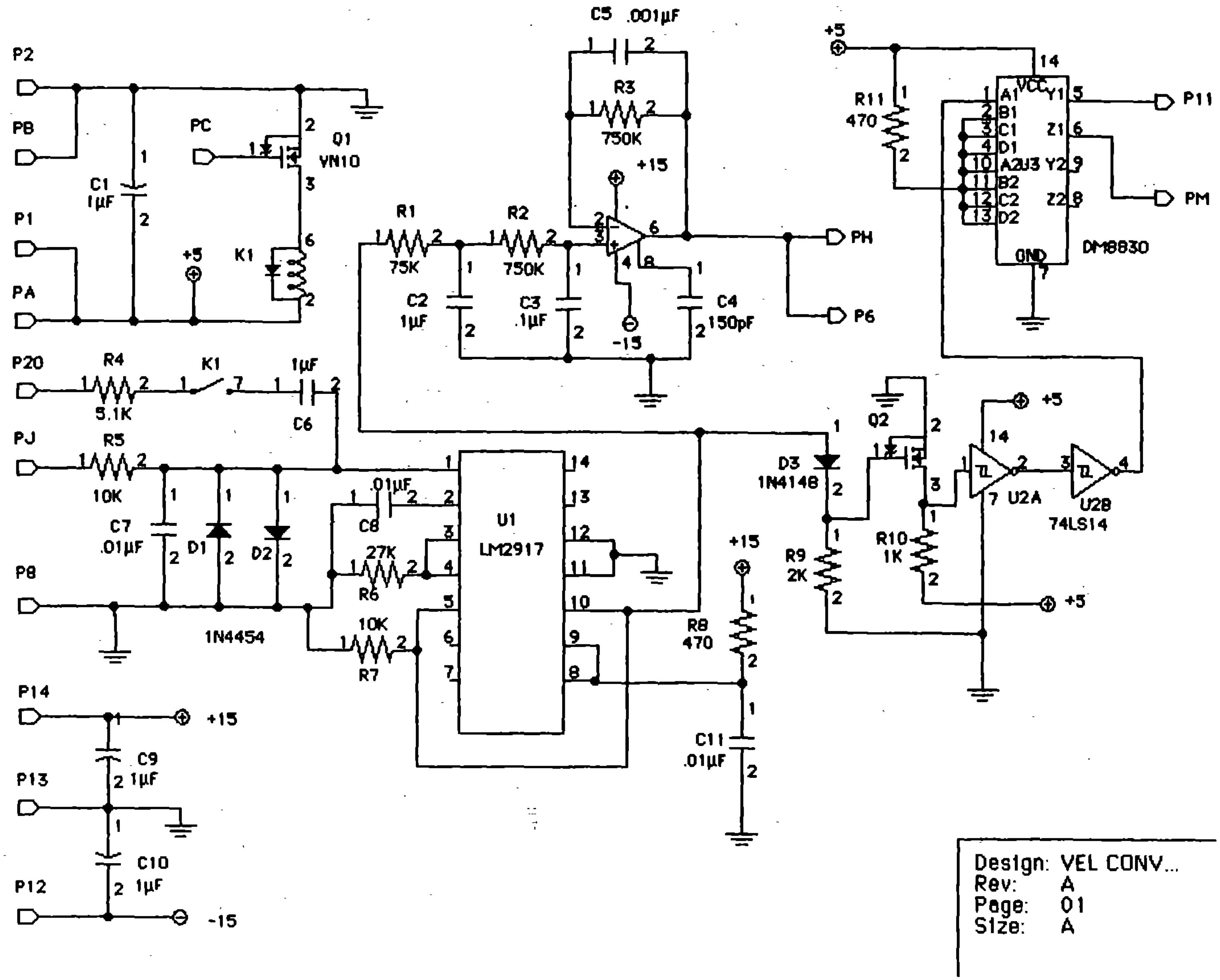


Figure 26. Schematic of the velocity-converter card.

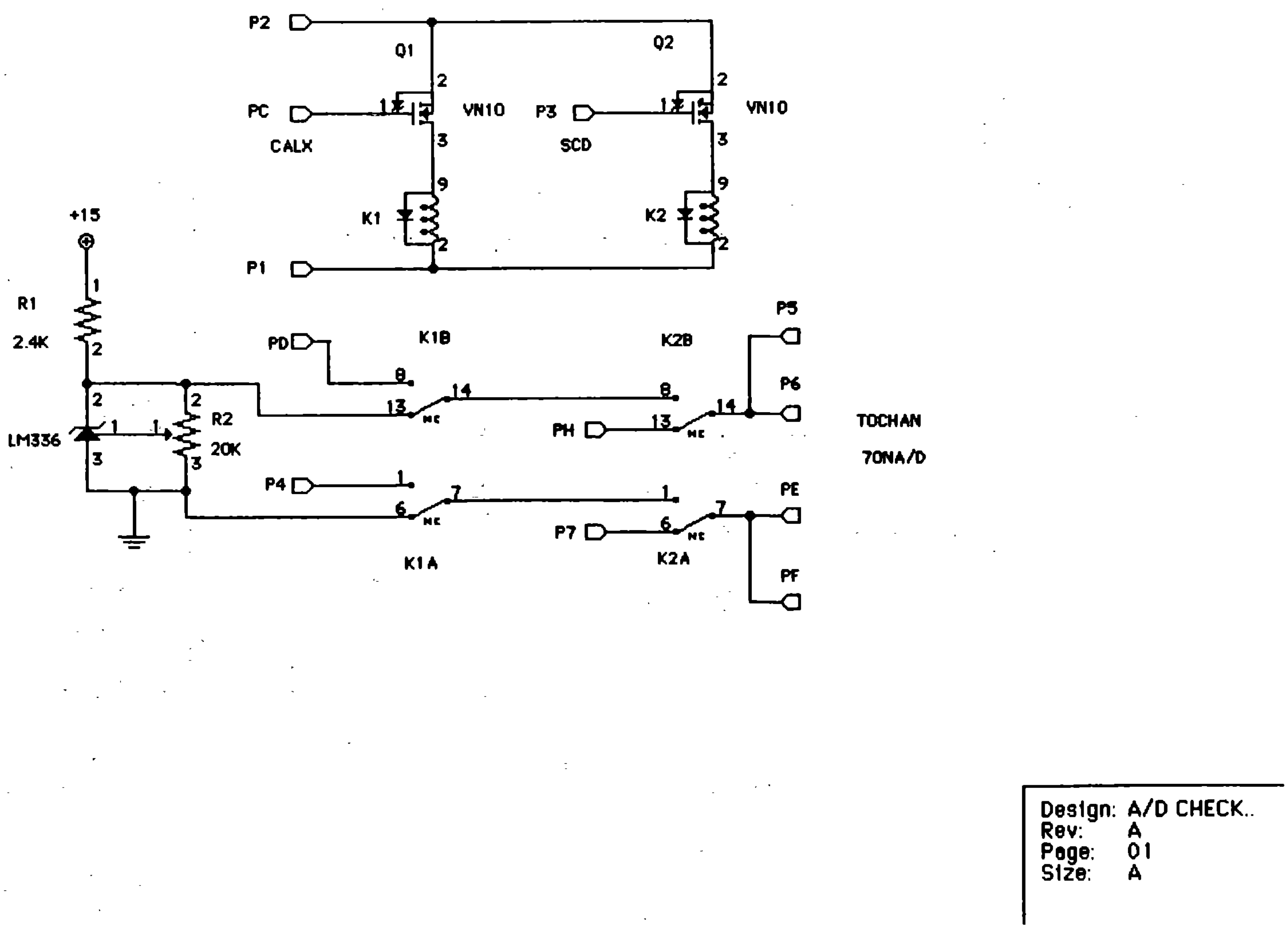


Figure 27. Schematic of the A/D check card.

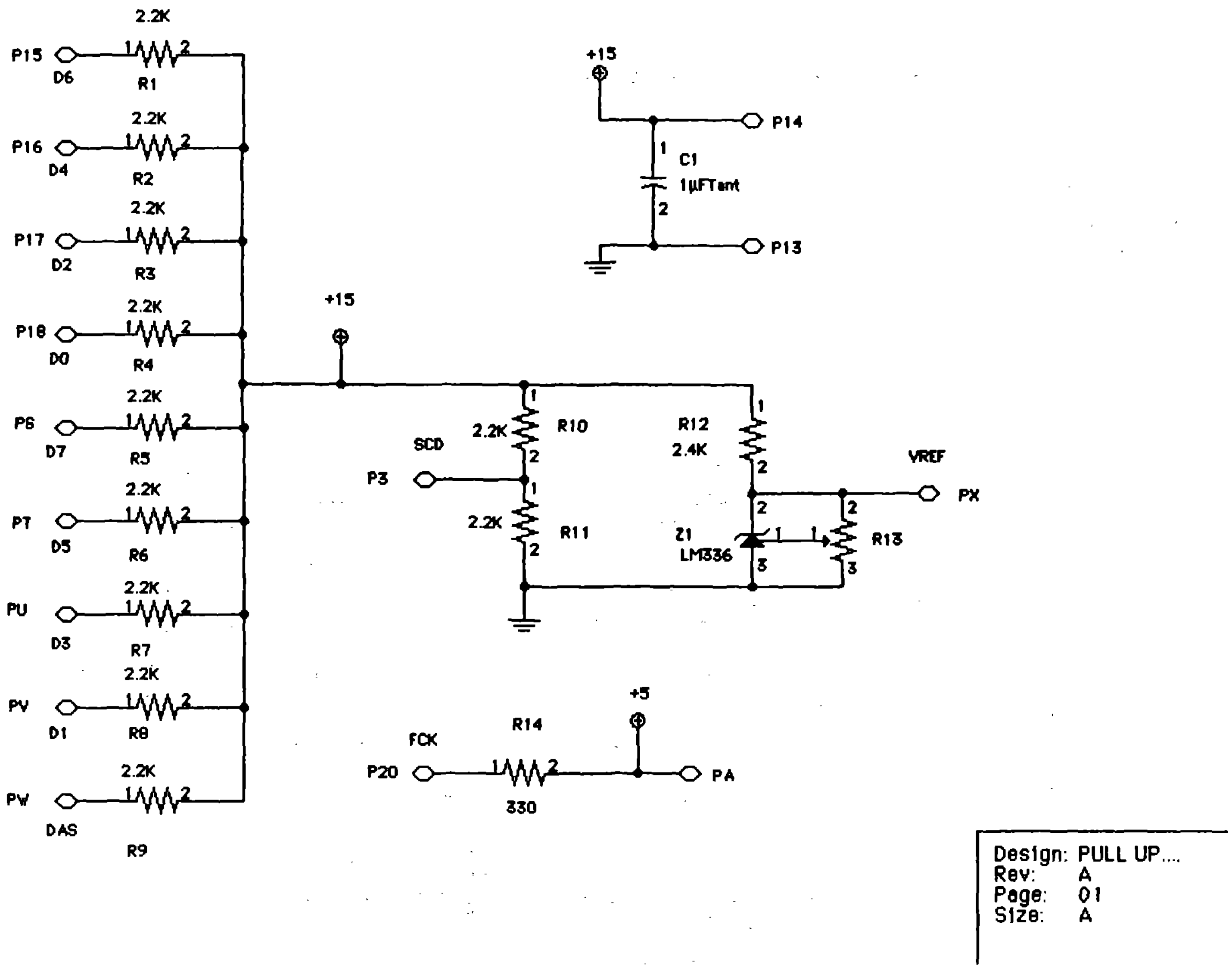


Figure 28. Schematic of the pull-up card.

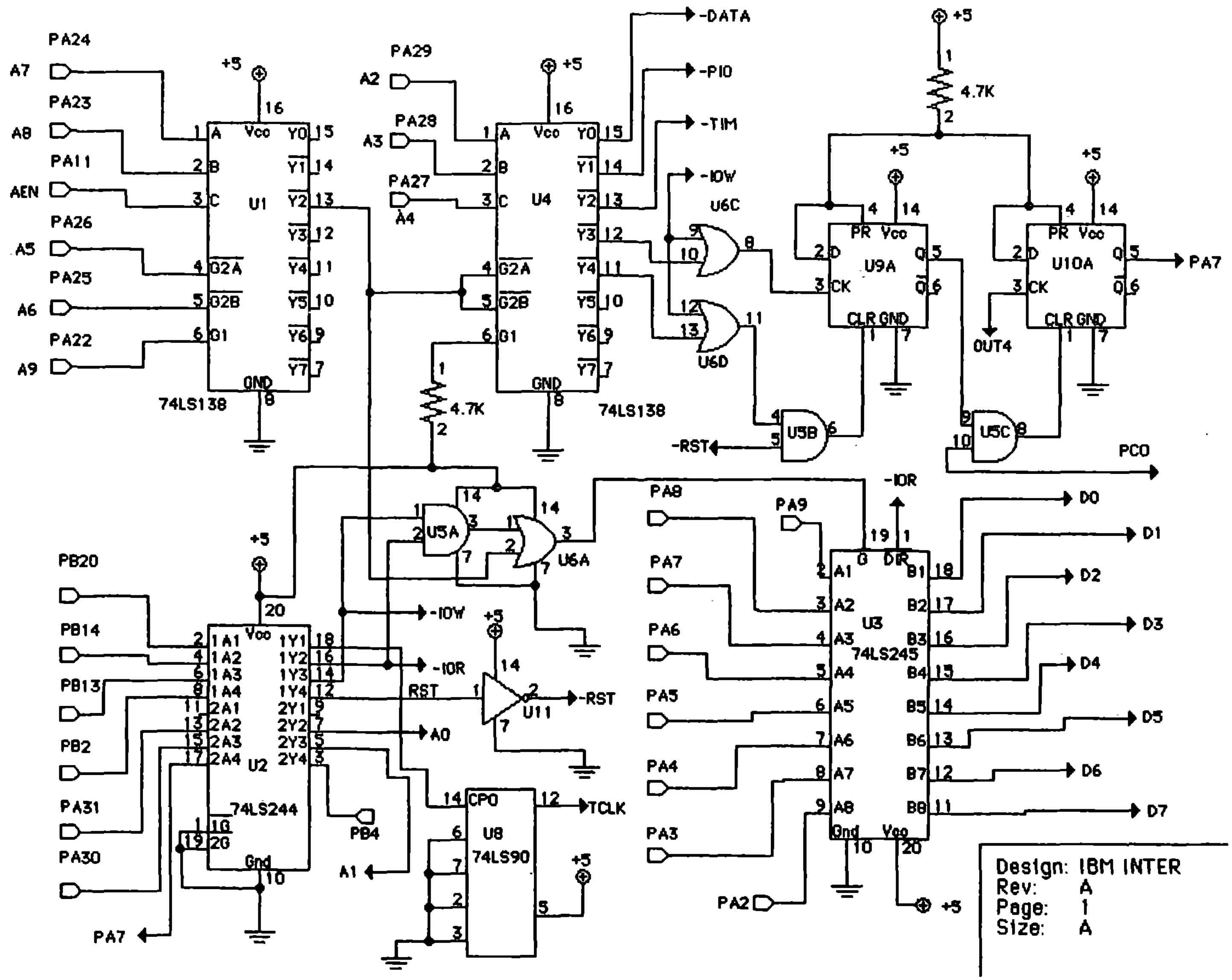


Figure 29. Schematic of the calibration IBM interface card (part 1 of 4).

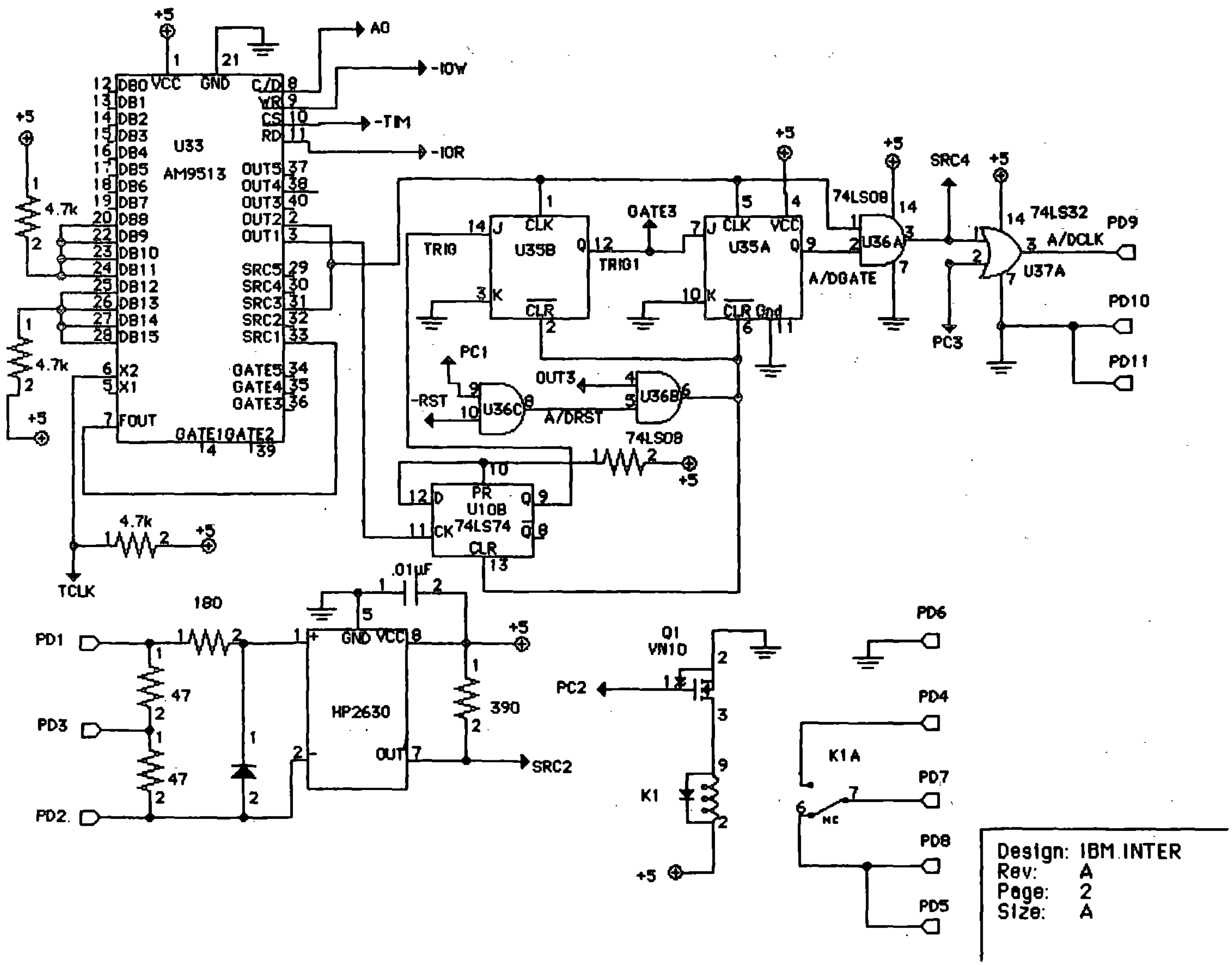


Figure 30. Schematic of the calibration IBM interface card (part 2 of 4).

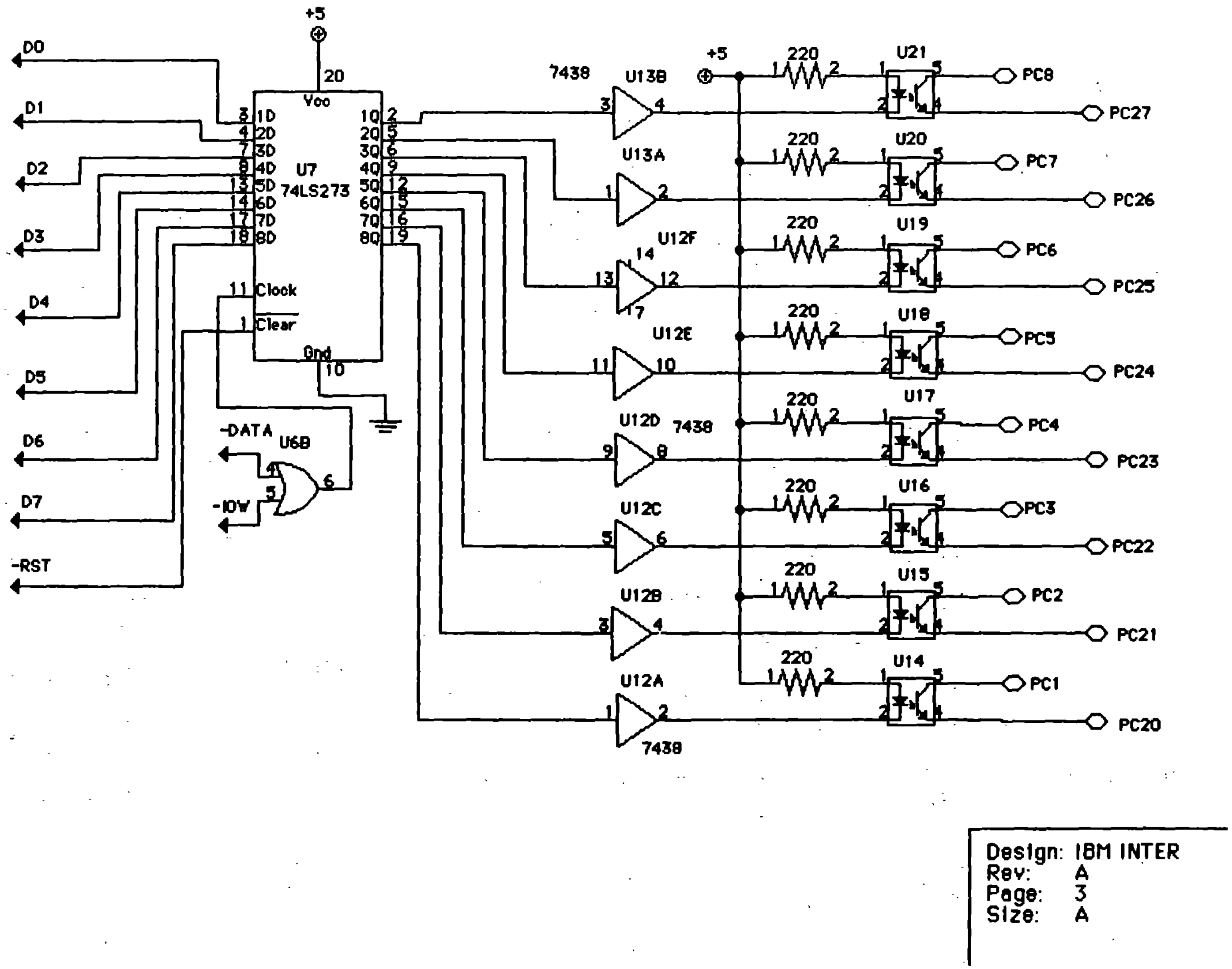


Figure 31. Schematic of the calibration IBM interface card (part 3 of 4).

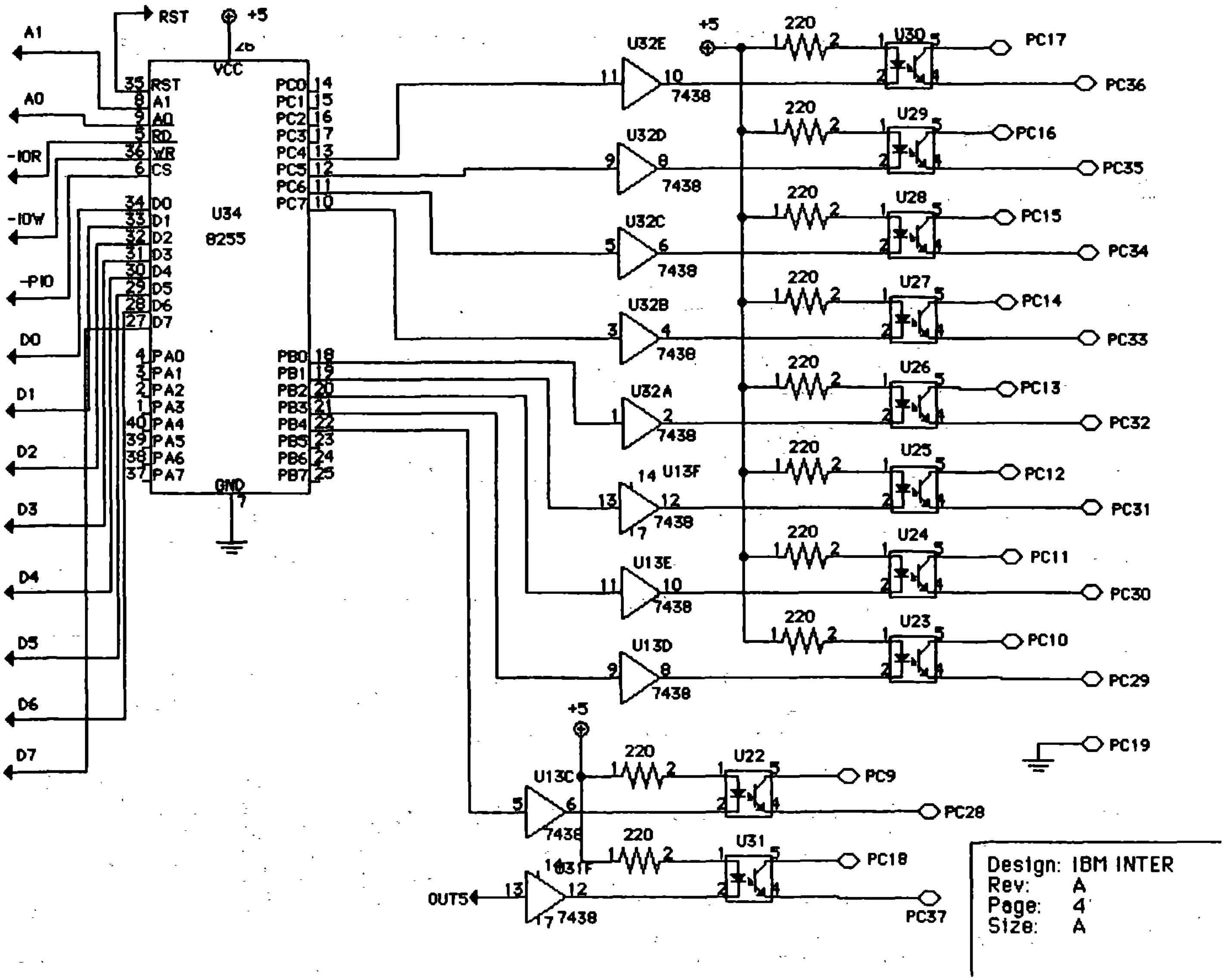


Figure 32. Schematic of the calibration IBM interface card (part 4 of 4).

APPENDIX B: CABLING INFORMATION

DB25-P-B to Analog Backplane	Function	Wire Color	DC37-P to IBM Interface
4	FCLK+	WHT	18
16	FCLK-	GRN	37
16			
5	DAS+	BLU	16
17	DAS-	BLK	35
17			
6	D0+	ORN	8
18	D0-	RED	27
18			
7	D1+	GRN	7
19	D1-	BLK	26
19			
8	D2+	BRN	6
20	D2-	RED	25
20			
9	D3+	YEL	5
21	D3-	RED	24
21			
10	D4+	RED	4
22	D4-	GRN	23
22			
11	D5+	BLU	3
23	D5-	RED	22
23			
12	D6+	WHT	2
24	D6-	RED	21
24			
13	D7+	WHT	1
25	D7-	BLK	20
25			

Figure 33. Cable diagram of the calibration control wiring (part 1 of 2).

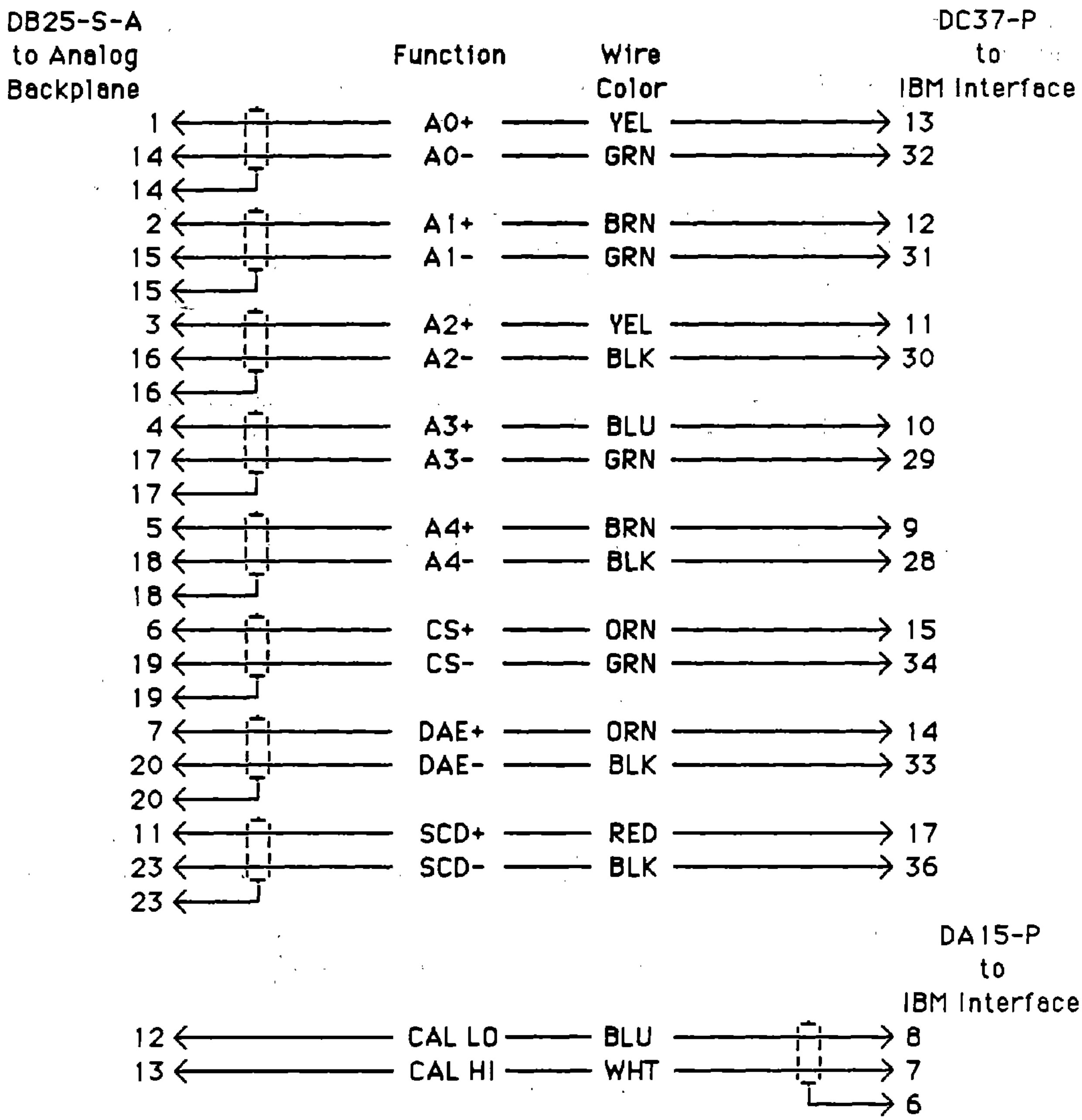


Figure 34. Cable diagram of the calibration control wiring (part 2 of 2).

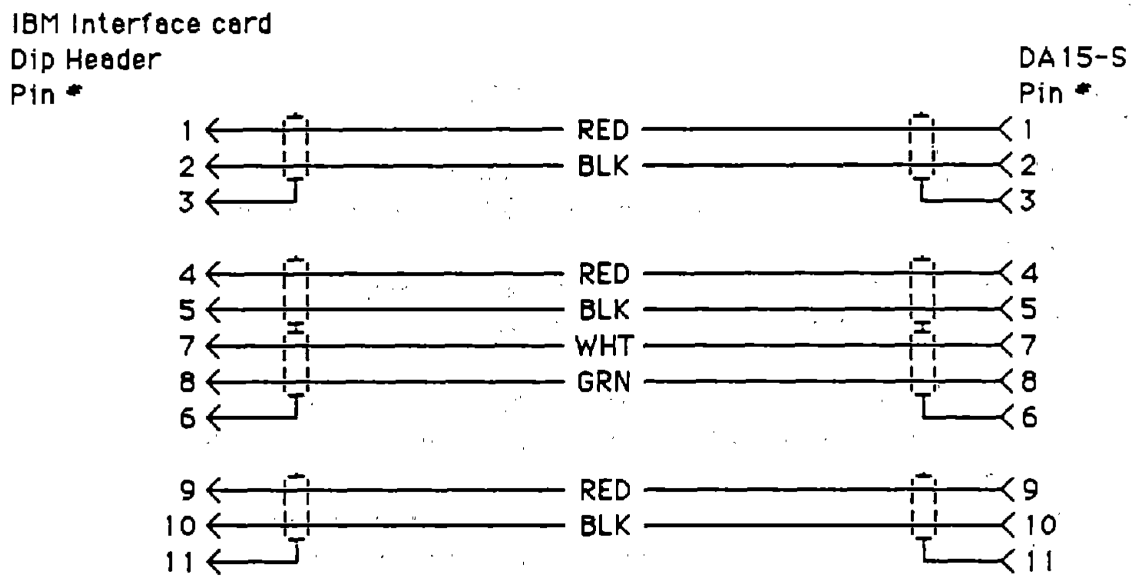


Figure 35. Cable diagram of the calibration IBM interface card DIP jumper wiring.

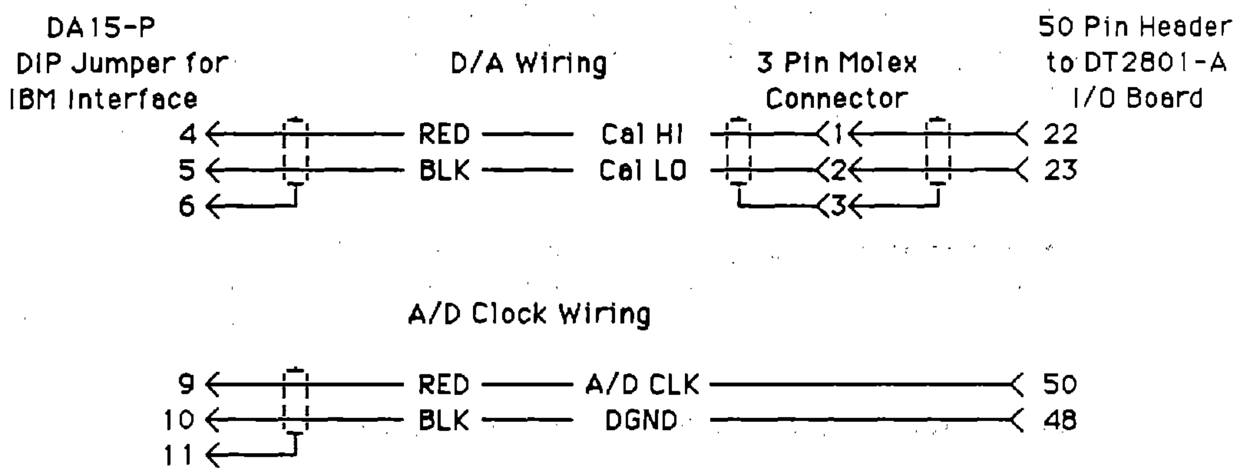


Figure 36. Cable diagram of the A/D clock and D/A wiring.

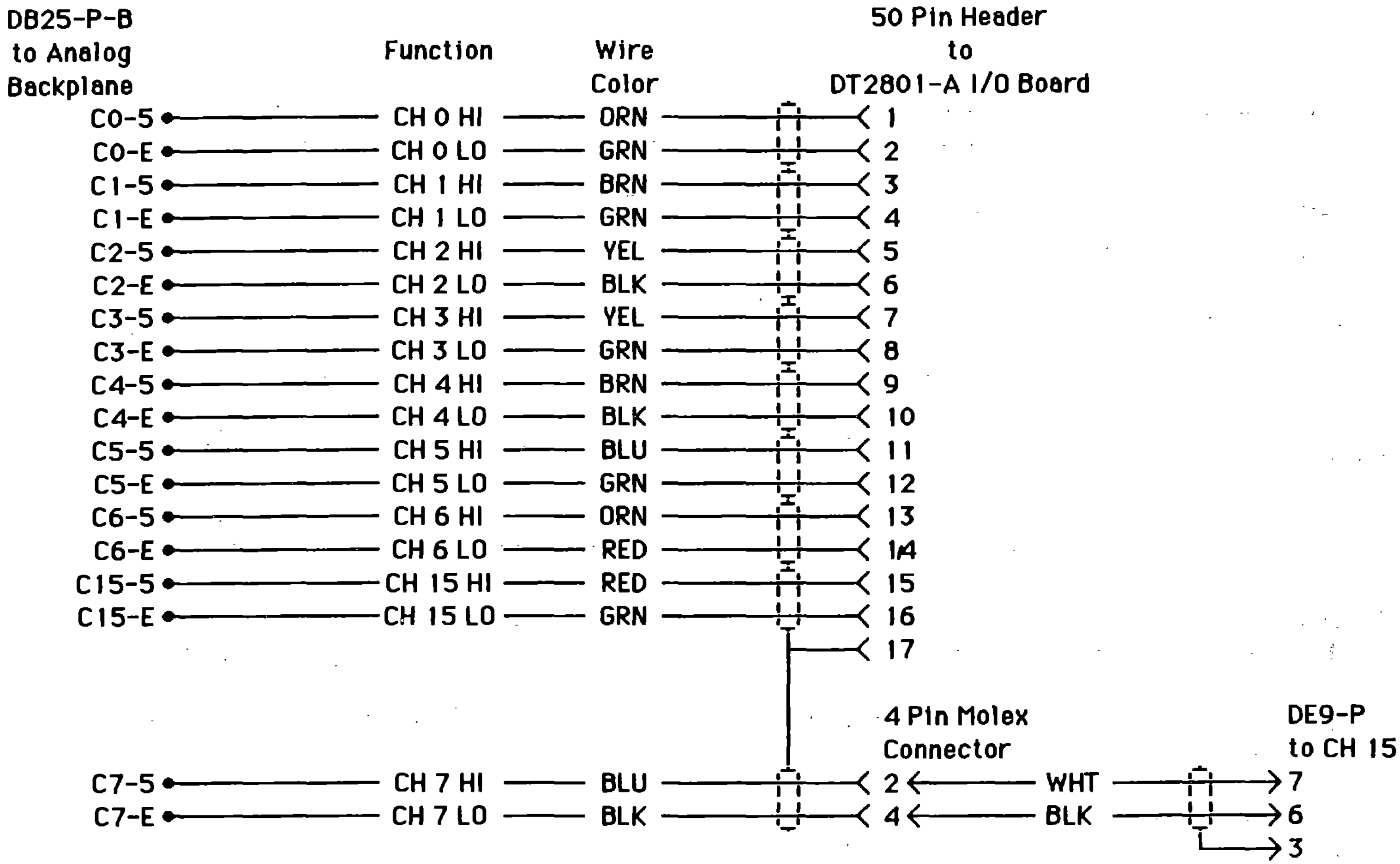


Figure 37. Cable diagram of the A/D wiring.

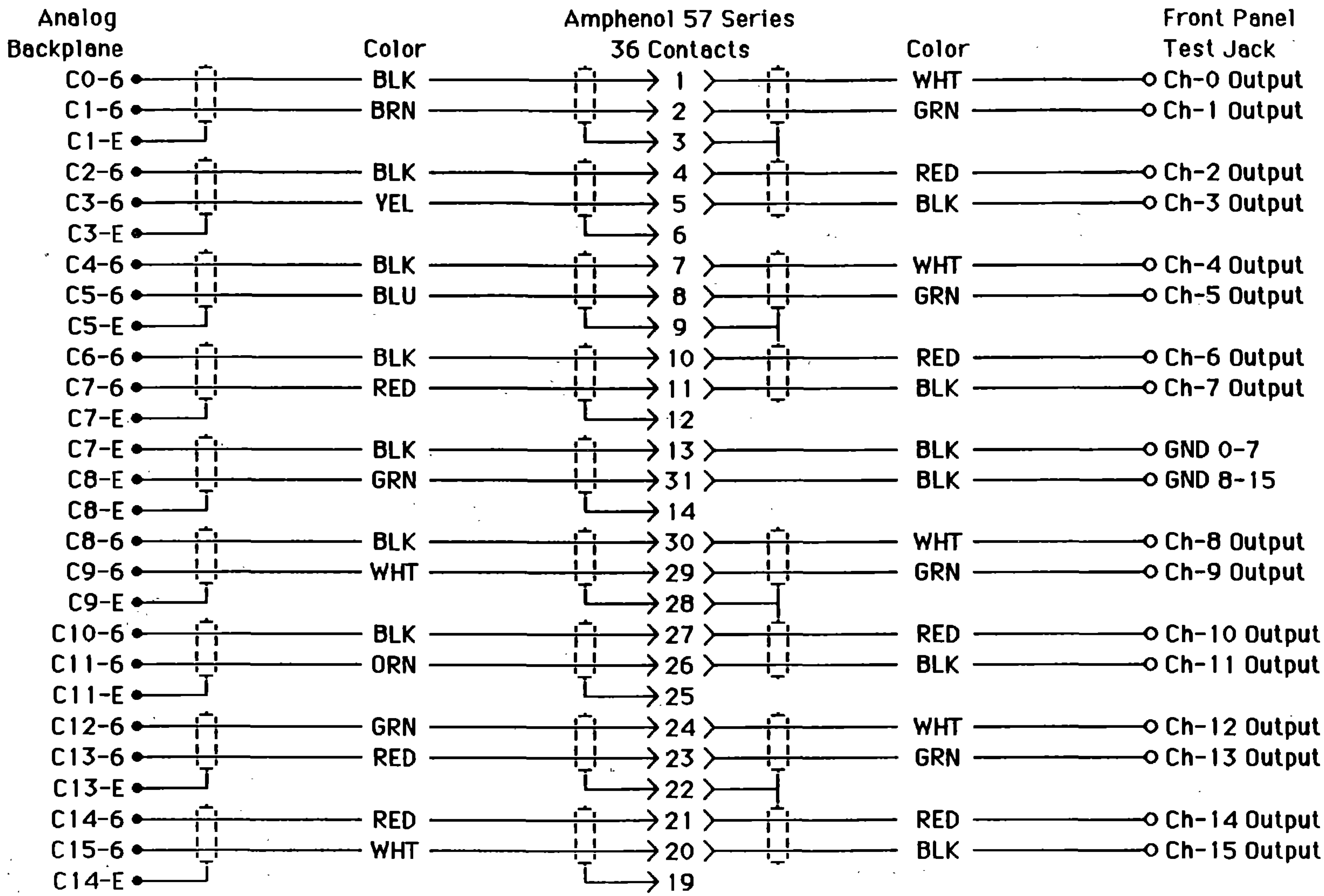


Figure 38. Cable diagram of the test jack wiring.

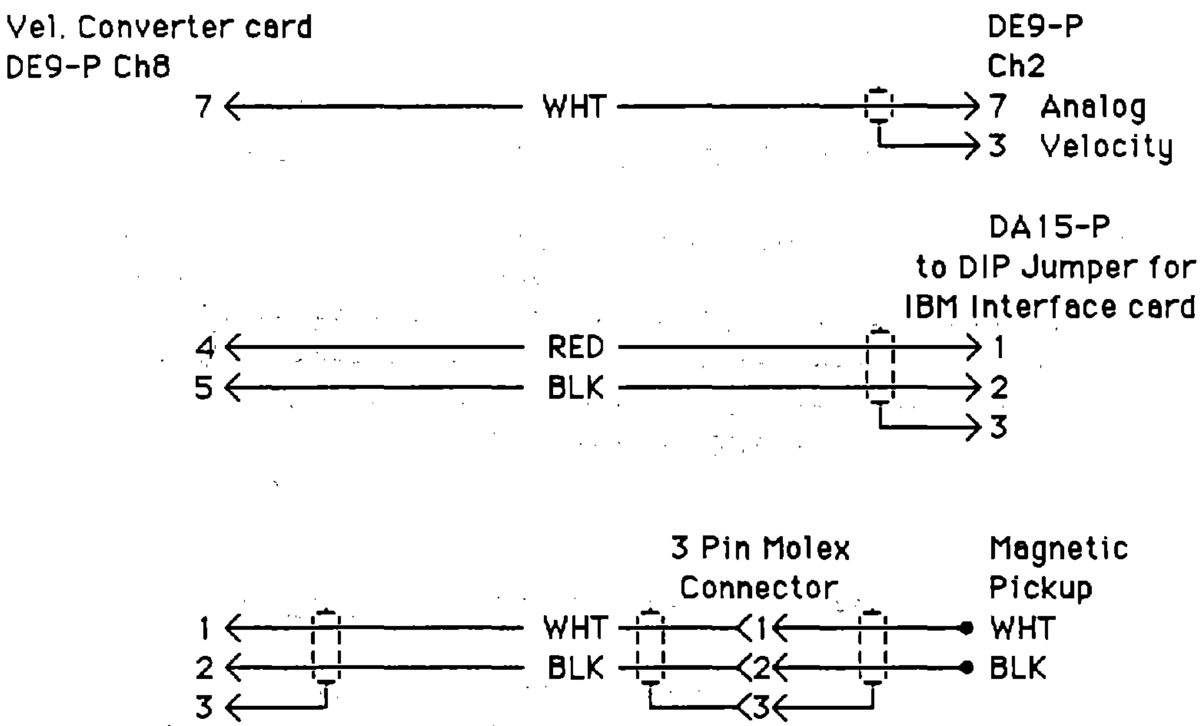


Figure 39. Cable diagram of the velocity wiring.

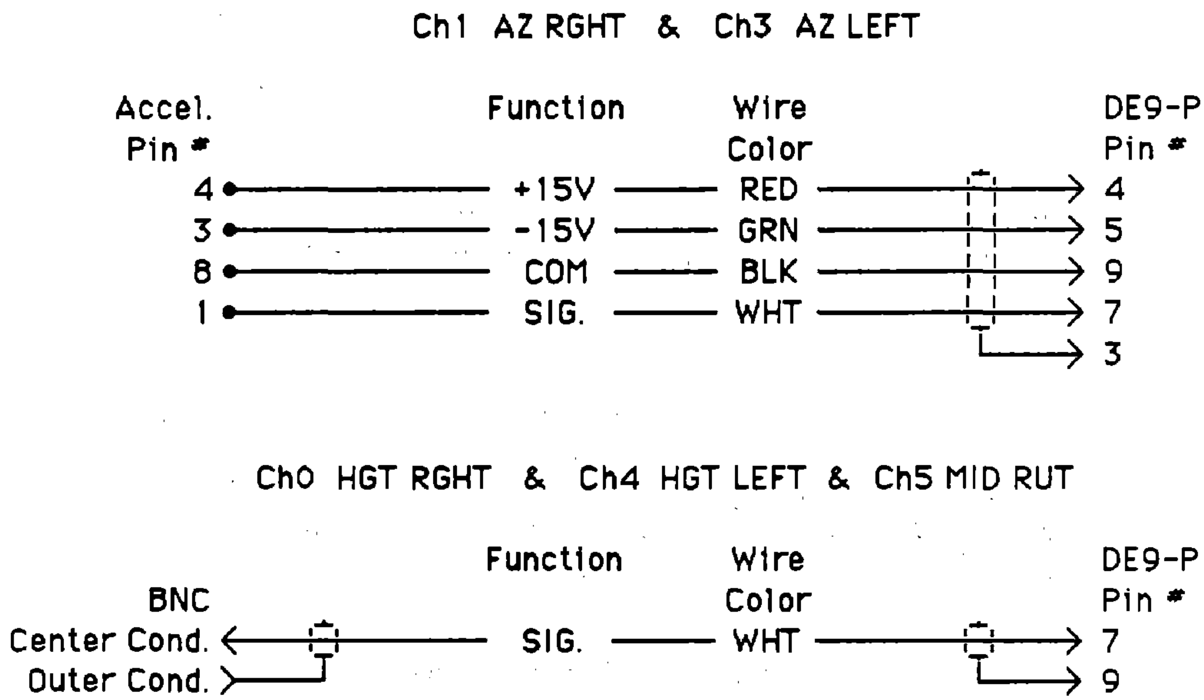


Figure 40. Cable diagram of the accelerometer and height sensor wiring.

APPENDIX C: FORTRAN EXTENSIONS

This appendix describes about 60 subroutines and functions that can be used with Fortran programs compiled for the IBM PC with the Microsoft compiler. They extend the standard Fortran language to allow closer interaction with the hardware of the IBM PC. Table 13 lists all of the routines for quick reference, and the remainder of this appendix describes the routines in more detail. The descriptions are divided into categories dealing with DOS file access, integer functions, screen I/O routines, "user friendly" input, and miscellaneous routines.

File Routines

This subsection describes the subroutines that allow the Fortran programmer to create, open, read, and write files directly through DOS function calls. These subroutines make reading and writing random access binary data files ten times faster than normal Microsoft Fortran I/O. This section also describes various Fortran subroutines that provide a user-friendly way to select file(s) and enter data.

Quick Reference

DFREE (DRIVE, CLUSA, CLUST, BYTES, SECTOR) — Get disk free space.
DRVSEL (DR) — Select a drive.
FINDF (FNAME, IERROR) — Find first matching file.
FINDN (IERROR) — Find next matching file.
FNMAKE (DR, NAME, EXT, FNAME, DIR) — Compose or decompose a filename.
FSEL (FNAME, ISEL, FILNA) — Select a file.
FSELALL (FNAME, ISEL, FILNA) — Select several files.
GETDTA (DTAAD) — Get disk transfer address.
GFILE (DR, FNAME, EXT, IEXIST, ROW, COL, IRET) — Get a filename.
HCLOSE (HANDLE, IER) — Close a file.
HCREAT (FNAME, HANDLE, IER) — Create a file.
HOPEN (FNAME, HANDLE, ACCESS, IER) — Open a file.
HPOS (HANDLE, METHOD, OFFSET, POINTER, IER) — Position a file.
HREAD (HANDLE, ARRAY, BYTES, RBYTES, IER) — Read from a file.
HWRITE (HANDLE, ARRAY, BYTES, RBYTES, IER) — Write to a file.
SETDA (DTAAD) — Set disk transfer address.

Table 13. Quick reference for the Fortran extensions library.

Each subprogram is a subroutine, unless it is specifically identified as a function.

ADDNUL (STRING, LEN) — Add a null to the end of string.
BEEP — Send a beep to the speaker.
BLANK (STRING, LEN) — Fill a string with blanks.
CLRLIN (L) — Clear line L.
CLRSCR — Clear the screen.
CONHEX (I, D) — Convert an integer into a hex string.
DFREE (DRIVE, CLUSA, CLUST, BYTES, SECTOR) — Get disk free space.
DRVSEL (DR) — Select a drive.
FINDF (FNAME, IERROR) — Find first matching file.
FINDN (IERROR) — Find next matching file.
FNMAKE (DR, NAME, EXT, FNAME, DIR) — Compose or decompose a filename.
FSEL (FNAME, ISEL, FILNA) — Select a file.
FSELALL (FNAME, ISEL, FILNA) — Select several files.
GCHAR (CHAR, ATT) — Get a character from the screen.
GCUR (ROW, COL) — Get the current cursor position.
GDATE (YEAR, MONTH, DAY) — Get the date.
GETDTA (DTAAD) — Get disk transfer address.
GETI (I, ILOW, IHIGH, ROW, COL, L, PFRMT, IRET) — Get an integer.
GETR (A, ALOW, AHIGH, ROW, COL, L, PFRMT, IRET) — Get a real number.
GETSTR (STRING, MAXLEN, ROW, COL, IRET) — Get a string.
GFILE (DR, FNAME, EXT, IEXIST, ROW, COL, IRET) — Get a filename.
GTIME (HOUR, MIN, SEC) — Get the time.
HCLOSE (HANDLE, IER) — Close a file.
HCREAT (FNAME, HANDLE, IER) — Create a file.
HOPEN (FNAME, HANDLE, ACCESS, IER) — Open a file.
HOWLNG (STRING, NTOTAL, LEN) — How long is the string.
HPOS (HANDLE, METHOD, OFFSET, POINTER, IER) — Position a file.
HREAD (HANDLE, ARRAY, BYTES, RBYTES, IER) — Read from a file.
HWRITE (HANDLE, ARRAY, BYTES, RBYTES, IER) — Write to a file.
Function IAND (J, K) — Bit-wise AND.
Function IGKEY () — Get a key press from keyboard.
INERROR (STRING, LEN) — Beep and write error message.
Function INOT (J) — Bit-wise complement.
Function INPB (J) — Input a byte.
Function INPW (J) — Input a word.
Function IOR (J, K) — Bitwise OR.
IOUTB (I, J) — Output a byte.
IOUTW (I, J) — Output a word.
Function IPEEKB (JJ) — Get a byte from memory.
Function IPEEKW (JJ) — Get a word from memory.
IPOKEB (I, JJ) — Put a byte into memory.
IPOKEW (I, JJ) — Put a word into memory.

Table 13. Quick reference for the Fortran extensions library (continued).

Function ISHFTL (J, COUNT) — Shift a word left.
Function ISHFTR (J, COUNT) — Shift a word right.
IVARPT (I, JJ) — Get the address of a variable.
Function IXOR (J, K) — Bit-wise exclusive OR.
KCLEAR — Clear the keyboard input buffer.
LJUST (STRING, LEN) — Left justify a string.
MENU (MNAME, MITEMS, MAVAIL, IDEF, IRET) — Select an item via a menu.
PCHAR (CHAR, ATT, COUNT) — Put a character on the screen
PHEX (I) — Print an integer in hexadecimal.
PHYSAD (N, JJ) — Calculate the physical address of a variable.
PWAIT (IP, N, M) — Wait for a condition on an input port.
RETPRO (IRET, I, J, IMAX, JMAX, PAGE, PAGMAX) — Process a return code.
SCRLDN (NLINES, UROW, UCOL, LROW, LCOL) — Scroll down a window.
SCRLUP (NLINES, UROW, UCOL, LROW, LCOL) — Scroll up a window.
SETCUR (ROW, COL) — Set the cursor position.
SETDA (DTAAD) — Set disk transfer address.
STRI (I, STRING, LEN) — Convert an integer into a string.
STRX (X, STRING, LEN) — Convert a real number into a string.
SUBNUL (STRING, LEN) — Remove a null from a string.
WAITKY — Wait for any keypress.
YESNO (I, ROW, COL, IRET) — Get a yes or no.

DFREE (DRIVE, CLUSA, CLUST, BYTES, SECTOR) *forext.obj*

This subroutine gets the amount of free space on a drive. The fraction of the drive that is available is CLUSA + CLUST. The number of free bytes on the drive is BYTES × SECTOR × CLUSA.

→ DRIVE	integer*2	drive number (0=default, 1=a, 2=b, etc.).
← CLUSA	integer*2	clusters available.
← CLUST	integer*2	total number of clusters on a drive.
← BYTES	integer*2	number of bytes per sector.
← SECTOR	integer*2	number of sectors per cluster.

DRVSEL (DR) *drvsel.obj*

Use a menu to select a drive (A-G).

↔ DR	char*1	on entry DR is the default drive (e.g., "C") and on exit it is the drive selected from the menu.

FINDF (FNAME, IERROR) *forext.obj*

Find the first file name that matches the one in FNAME and place the name in the disk transfer area starting at the 31-st byte. This should be preceded by a call to GETDTA to get and save the current disk transfer address and then a call to SETDTA to set the disk transfer area to a character*1 variable.

→ FNAME	char* (*)	ASCIIZ filename that begins with the drive specifier and can contain the wild card characters *?* and *.
← IERROR	integer*2	error return with 0=no error and 18=no matching filename.

FINDN (IERROR) *forext.obj*

Find the next matching filename after a call to FINDF. The subroutine FINDF is only called once to get the first filename and then FINDN is called repeatedly until there are no more matching files.

← IERROR	integer*2	error return with 0=no error and 18=no more files.

FNMAKE (DR, NAME, EXT, FNAME, DIR) *fnmake.obj*

If DIR is equal to zero then set FNAME=DR:NAME.EXT removing all embedded spaces. For example, if DR="A", NAME="FILE", and EXT="OBJ" then FNAME would be set to "A:FILE.OBJ". If DIR is equal to one, then take FNAME and decompose it into its three parts DR, NAME, and EXT. For example, if FNAME="c:myfile.dt" then on output DR="c", NAME="myfile", and EXT="dt".

↔ DR	char*1	one letter drive specification.
↔ NAME	char*8	eight letter file name.
↔ EXT	char*3	three letter extension.
↔ FNAME	char*16	sixteen letter file specification DR:NAME.EXT.
→ DIR	integer*2	if zero then compose file name; if one decompose file name.

FSEL (FNAME, ISEL, FILNA) *fsel.obj*

This subroutine allows a user to select a file for some operation without having to type it in. The calling program typically calls DRVSEL before FSEL to get the drive of interest and then creates a file specification that consists of the drive , name, and extension that can include wild card characters. For example, if the program needs a data file to read and all data files end with the extension "dta" then the calling program would set FNAME="a:*.dta". Up to four screens (twenty rows by four columns of filenames) are then displayed and the user selects a file via the cursor keys. The filenames are put into reverse video as they are selected. The right and left arrow keys move across the four columns and the up and down arrows move up and down the rows. The page-up key is used to see the next screen of files while the page-down key allows the viewing of the previous screen. When the file that the user wants is in reverse video, it is selected by pressing the end key. The selected filename is put into FILNA and ISEL is set to one. If no files are selected, then ISEL will be set to zero.

→ FNAME	char*15	15 character filename that can include wild-card characters.
← ISEL	integer*2	the number of files selcted by the user (either 0 or 1).
→ FILNA	any	any array of at least 3328 bytes which is over-written with up to 256 filenames.

FSELALL (FNAME, ISEL, FILNA) *fselall.obj*

This subroutine allows the user to select up to 80 files at a time. It is typically called when some form of batch processing is to be done. Before FSELALL is called, the program should get a file specification from the user and put it into FNAME. For example, if the user wants to select all files beginning with "my" he could enter "my*.*". This would be passed through FNAME and FSELALL would put the list of all files matching this specification on the screen. The user is asked to verify that all the files are to be selected and if a yes is answered then the array FILNA is filled with the names and ISEL will reflect the number of files selected.

→ FNAME	char*15	15 character filename that can include wild card characters.
↔ ISEL	integer*2	on input the maximum number of files that can be selcted by the user (<= 80) and on output the actual number of files selected.
↔ FILNA	char*16	the character array that receives the filenames.

GETDTA (DTAAD) *forext.obj*

This subroutine gets the current disk tranfer area address and puts its offset and segment into the array DTAAD. This routine is used to save this address before any calls to FINF or FINDN are done to ensure no files will be corrupted.

←	DTAAD	integer*2	array that will contain the address of the disk transfer area where DTAAD (1)=offset and DTAAD (2)=segment.

GFILE (DR, FNAME, EXT, IEXIST, ROW, COL, IRET) *utils.obj*

This subroutine prints the default file name FNAME to the screen at the position (ROW, COL), allows the user to edit this name (change or enter a new name), checks the name for illegal characters, and finds out if the file already exists. Note, only the eight-character name part of the file name can be edited.

→	DR	char*1	drive letter, e.g., *A*.
↔	FNAME	char*8	on input, this is the default file name. On output, it is the name typed by the user.
→	EXT	char*3	file extension.
←	IEXIST	logical*2	.TRUE. if file DR:FNAME.EXT already exists; otherwise .FALSE.
→	ROW	integer*2	row address where FNAME is to be written.
→	COL	integer*2	column address where FNAME is to be written.

HCLOSE (HANDLE, IER) *forext.obj*

Close a file that was opened by HOPEN or created and opened by a call to HCREAT.

→	HANDLE	integer*2	file handle assigned by DOS on HOPEN or HCREAT.
←	IER	integer*2	error return 0=no error.

HCREAT (FNAME, HANDLE, IER) *forext.obj*

Create a file named FNAME and have DOS refer to it by HANDLE. If the file already exists, then an error will be returned.

→	FNAME	char**	ASCIIZ filename.
←	HANDLE	integer*2	file handle assigned to FNAME by DOS.
←	IER	integer*2	error return 0=no error.

HOPEN (FNAME, HANDLE, ACCESS, IER) *forext.obj*

Open a file named FNAME and have DOS refer to it by HANDLE. If the file does not exist, then an error will be returned.The file can be opened as read only, write only, or as both read and write.

→	FNAME	char**	ASCIIZ filename.

←	HANDLE	integer*2	file handle assigned to FNAME by DOS.
→	ACCESS	integer*2	file access code 0=read only, 1=write, 2=read and write.
←	IER	integer*2	error return 0=no error.

HPOS (HANDLE, METHOD, OFFSET, POINTER, IER) *forext.obj*

Position a file pointer referred to by HANDLE at OFFSET number of bytes from either the beginning of the file, from the current pointer, or from the end of the file. This subroutine allows random accessing of disk files. For example, if one wanted to skip over the first 2048 bytes of a file and then read the next 1024, HPOS would be called after HOPEN with METHOD=0 and OFFSET=2048. Any subsequent reads or writes would occur from this pointer onward.

→	HANDLE	integer*2	file handle assigned by DOS on HOPEN or HCLOSE.
→	METHOD	integer*2	method of positioning 0=absolute positioning from the beginning of the file, 1=relative positioning from the current position, and 2=from the end of the file.
→	OFFSET	integer*4	offset into the file.
←	POINTER	integer*4	returned file pointer.
←	IER	integer*2	error return 0=no error.

HREAD (HANDLE, ARRAY, BYTES, RBYTES, IER) *forext.obj*

Read from a file referred to by HANDLE and put BYTES number of bytes into the buffer ARRAY.

→	HANDLE	integer*2	file handle assigned by DOS on HOPEN or HCLOSE.
←	ARRAY	any	array into which the file is to be read.
→	BYTES	integer*4	number of bytes to read (< 65535).
←	RBYTES	integer*4	number of bytes actually read.
←	IER	integer*2	error return 0=no error.

HWRITE (HANDLE, ARRAY, BYTES, RBYTES, IER) *forext.obj*

Write BYTES number of bytes from the buffer ARRAY into the file referred to by HANDLE.

→	HANDLE	integer*2	file handle assigned by DOS on HOPEN or HCLOSE.
→	ARRAY	any	array containing stuff to write.
→	BYTES	integer*4	number of bytes to write (< 65535).
←	RBYTES	integer*4	number of bytes actually written.
←	IER	intger*2	error return 0=no error.

SETDA (DTAAD) *forext.obj*

Set the disk transfer area address to the offset contained in DTAAD (1) and to the segment contained in DTAAD (2). This address is either one gotten previously by a call to GETDA or the address of a character variable found via a call to IVARPT.

→ DTAAD	integer*2	array that contains the address of the disk transfer area where DTAAD (1)=offset and DTAAD (2)=segment.

Integer Functions

This subsection describes a set of integer*2 functions that add bit-wise logical functions to Fortran 77. These functions are different than the normal Fortran logical operators .OR., .AND., and .NOT. in that they perform the logical operation on every bit of the operand(s). In addition to the logical functions, there are functions similar to the Basic functions of PEEK and INP which return word as well as byte values. In the examples below the # is used to indicate a hexadecimal number.

Quick Reference

IAND (J, K) — Bit-wise AND.
IGKEY () — Get a key press from keyboard.
INOT (J) — Bit-wise complement.
INPB (J) — Input a byte.
INPW (J) — Input a word.
IOR (J, K) — Bitwise OR.
IPEEKB (JJ) — Get a byte from memory.
IPEEKW (JJ) — Get a word from memory.
ISHFTL (J, COUNT) — Shift a word left.
ISHFTR (J, COUNT) — Shift a word right.
IXOR (J, K) — Bit-wise exclusive OR.

Subroutine Descriptions

IAND (J, K) *forext.obj*

Return the bit-wise AND of the two arguments. For example, IAND (#AAAA, #5555) = 0 and IAND (#FF00 , #AAAA) = #AA00.

→ J	integer*2	first argument.
→ K	integer*2	second argument.

IGKEY () *forext.obj*

Get a keypress from the keyboard. If there was no key pressed since the last call to KCLEAR then this function returns a zero. If there is a character in the keyboard buffer, then a call to IGKEY will return it. If the value returned is greater than zero, then the key that was pressed corresponds to a normal ASCII character (between 0 and 128). If the value is less than zero, then the key that was pressed corresponds to an extended key and the absolute value of the returned number will indicate which key was pressed (e.g., if IGKEY ()=-72, then the key was the extended key 72 or an up arrow). This function is used in all of the user-friendly input routines to overcome the limitations of Fortran input.

INOT (J) *forext.obj*

Return the one's complement of the argument. For example, INOT (#AAAA) = #5555.

→ J integer*2 number to complement.

IOR (J, K) *forext.obj*

Return the bit-wise OR of the two arguments. For example, IOR (#AAAA , #5555)=#FFFF.

→ J integer*2 first argument.
→ K integer*2 second argument.

INPB (J) *forext.obj*

Get a byte from the I/O port addressed by J.

→ J integer*2 port address.

INPW (J) *forext.obj*

Get a word from the I/O port addressed by J.

→ J integer*2 port address.

IPEEKB (JJ) *forext.obj*

Get the byte in memory addressed by the array JJ.

→ JJ integer*2 memory address where JJ (1)= offset and JJ (2)=segment.

IPEEKW (JJ) *forext.obj*

Get the word in memory addressed by the array JJ.

→ JJ integer*2 memory address where JJ (1)= offset and JJ (2)=segment.

ISHFTL (J, COUNT) *forext.obj*

Shift the number in J, COUNT bits to the left. Fill the bits shifted in with zero. For example, ISHFTL (#FFFF , 8) = #FF00 and ISHFTL (#6789 , 4) = #7890.

→ J	integer*2	number to be shifted.
→ COUNT	integer*2	number of bit positions to shift.

ISHFTR (J, COUNT) *forext.obj*

Shift the number in J, COUNT bits to the right. Fill the bits shifted in with zero. For example, ISHFTR (#FFFF , 8) = #00FF and ISHFTR (#6789 , 4) = #0678.

→ J	integer*2	number to be shifted.
→ COUNT	integer*2	number of bit positions to shift.

IXOR (J, K) *forext.obj*

Return the bit-wise exclusive OR of the two arguments. For example, IXOR (#AAAA , #5555) = #FFFF and IXOR (#AAAA , #FFFF) = #5555.

→ J	integer*2	first argument.
→ K	integer*2	second argument.

Screen I/O Routines

The following section describes the subroutines that allow the Fortran program to interface directly with the BIOS screen routines. The user-friendly input routines employ these routines to circumvent the teletype kind of I/O normally associated with Fortran.

Quick Reference

BEEP — Send a beep to the speaker.
CLRLIN (L) — Clear line L.
CLRSCR — Clear the screen
GCHAR (CHAR, ATT) — Get a character from the screen.
GCUR (ROW, COL) — Get the current cursor position.
KCLEAR — Clear the keyboard input buffer.
PCHAR (CHAR, ATT, COUNT) — Put a character on the screen.
SCRLDN (NLINES, UROW, UCOL, LROW, LCOL) — Scroll down a window.
SCRLUP (NLINES, UROW, UCOL, LROW, LCOL) — Scroll up a window.
SETCUR (ROW, COL) — Set the cursor position.

BEEP *utils.obj*

Send a beep to the speaker (same as a bell on a teletype, i.e., control G). The beep normally tells the user that he has done something inappropriate.

CLRLIN (L) *utils.obj*

Clear the line L on the screen.

→ L	integer*2	line of the screen to clear $0 \le L \le 24$ (line 0 is at the top of the screen).

CLRSCR *utils.obj*

Clear the entire screen.

GCHAR (CHAR, ATT) *forext.obj*

Get a character and its attribute from the current cursor location.

← CHAR	char*1	character received from the screen
← ATT	integer*2	attribute of the character where 7=normal video, 112=reverse video, =blinking, =intensified.

GCUR (ROW, COL) *forext.obj*

Get the current cursor position (the position (0, 0) is at the top left corner of the screen).

← ROW	integer*2	row address of the cursor where $0 \le ROW \le 24$.
← COL	integer*2	column address of the cursor where $0 \le COL \le 79$.

KCLEAR *utils.obj*

Clear the keyboard buffer. This subroutine should be called before a reference to IGKEY if any keys already in the buffer are to be ignored.

PCHAR (CHAR, ATT, COUNT) *forext.obj*

Put a character with the attribute ATT to the screen at the current cursor position. If COUNT is greater than one, then copy the character COUNT times. For example, to put a line of hyphens across the screen CALL PCHAR ("-", 7, 80).

→ CHAR	char*1	character to put to the screen.
→ ATT	integer*2	attribute of the character where 7=normal video, 112=reverse video, =blinking, =intensified.
→ COUNT	integer*2	number of times to repeat character.

SCRLDN (NLINES, UROW, UCOL, LROW, LCOL) *forext.obj*

Scroll the window whose upper left corner is (UROW, UCOL) and whose lower right corner is (LROW, LCOL) NLINES lines down.

→ NLINES	integer*2	number of lines to scroll.
→ UROW	integer*2	row coordinate of the upper left corner of the window.
→ UCOL	integer*2	column coordinate of the upper left corner of the window.
→ LROW	integer*2	row coordinate of the lower right corner of the window.
→ LCOL	integer*2	column coordinate of the lower right corner of the window.

SCRLUP (NLINES, UROW, UCOL, LROW, LCOL) *forext.obj*

Scroll the window whose upper left corner is (UROW, UCOL) and whose lower right corner is (LROW, LCOL) NLINES lines up.

→ NLINES	integer*2	number of lines to scroll.
→ UROW	integer*2	row coordinate of the upper left corner of the window.
→ UCOL	integer*2	column coordinate of the upper left corner of the window.
→ LROW	integer*2	row coordinate of the lower right corner of the window.
→ LCOL	integer*2	column coordinate of the lower right corner of the window.

SETCUR (ROW, COL) *forext.obj*

Set the cursor to the position indicated (the position (0, 0) is at the top left corner of the screen). For Fortran output to be correctly placed on the screen, the format character "\" should be used as the last character in the format string. For example,

```
WRITE (*, '(I4\)') I
```

→ ROW	integer*2	row address to place the cursor where $0 \leq ROW \leq 24$.
→ COL	integer*2	column address to place the cursor where $0 \leq COL \leq 79$.

User-Friendly Input

The following routines were written to give a user of a Fortran program a consistent and friendly way to input numbers and strings and to select one from many options via a menu.

The routines GETI, GETR, GETSTR, and YESNO all operate in a similar manner. First a default value is printed and the field is highlighted in reverse video. Thus, the user immediately knows what kind of input is expected and how many characters he can type in.

The field can be edited by using the left and right arrows to move the cursor to any character in the field and then overtyping. When the user is happy with the entry, any one of eight terminating keys is pressed. These keys are *Return*, up arrow, down arrow, page up, page down, control left arrow, control right arrow, and *End*. In the simple case of a single input, these keys all result in the same action. For page editing, however, they allow the user to edit many fields displayed on the screen by moving up, down, to the next or previous page, left, and right. The *End* key always terminates an input session. The default value can be accepted immediately by hitting any of the terminating keys. If the user input is not within a specified range or if illegal characters are entered, these routines will notify the user and replace what was typed in by the default value. Thus Fortran runtime input errors can be eliminated.

Quick Reference

GETI (I, ILOW, IHIGH, ROW, COL, L, PFRMT, IRET) — Get an integer.
GETR (A, ALOW, AHIGH, ROW, COL, L, PFRMT, IRET) — Get a real number.
GETSTR (STRING, MAXLEN, ROW, COL, IRET) — Get a string.
INERROR (STRING, LEN) — Beep and write error message.
MENU (MNAME, MITEMS, MAVAIL, IDEF, IRET) — Select an item via a menu.
YESNO (I, ROW, COL, IRET) — Get a yes or no.
WAITKY — Wait for any keypress.

Subroutine Descriptions

GETI (I, ILOW, IHIGH, ROW, COL, L, PFRMT, IRET) *util3.obj*

Print the default integer to the screen at (ROW, COL) and allow editing of the field of width L. Accept only integers that are greater or equal to ILOW and less than or equal to IHIGH.

↔ I	integer*2	the default integer on input and the number supplied by the user on output.
→ ILOW	integer*2	the lower limit for I.
→ IHIGH	integer*2	the upper limit for I.
→ ROW	integer*2	row address of the start of the edit field $(0 \leq ROW \leq 24)$.
→ COL	integer*2	column address of the start of the edit field $(0 \leq COL \leq 79)$.
→ L	integer *2	width of the field.
→ PFRMT	char	format to print the number (width = L) e.g. ' (I4\)'.
← IRET	integer*2	Terminating output code where 0=carriage return, 1=up arrow, 2=down arrow, 3=page up, 4=page down, 5=ctrl left arrow, 6= ctrl right arrow, and 9=end.

GETR (A, ALOW, AHIGH, ROW, COL, L, PFRMT, IRET) *util3.obj*

Print the default real number to the screen at (ROW, COL) and allow editing of the field of width L. Accept only numbers that are greater or equal to ALOW and less than or equal to AHIGH.

↔	A	real*4	the default number on input and the number supplied by the user on output.
→	ALOW	real*4	the lower limit for A.
→	AHIGH	real *4	the upper limit for A.
→	ROW	integer*2	row address of the start of the edit field ($0 \leq ROW \leq 24$).
→	COL	integer*2	column address of the start of the edit field ($0 \leq COL \leq 79$).
→	L	integer *2	width of the field.
→	PFRMT	char	format to print the number (width = L) e.g. ' (F8.2\)'.
←	IRET	integer*2	Terminating output code where 0=carriage return, 1=up arrow, 2=down arrow, 3=page up, 4=page down, 5=ctrl left arrow, 6= ctrl right arrow, and 9=end.

GETSTR (STRING, MAXLEN, ROW, COL, IRET) *getstrng.obj*

Highlight the string of length MAXLEN in reverse video and allow editing of the field. Note this routine does not write the default to the screen. It assumes that it has been already written.

←	STRING	char	the output string.
→	MAXLEN	integer*2	length of the edit field.
→	ROW	integer*2	row address of the start of the edit field ($0 \leq ROW \leq 24$).
→	COL	integer*2	column address of the start of the edit field ($0 \leq COL \leq 79$).
←	IRET	integer*2	Terminating output code where 0=carriage return, 1=up arrow, 2=down arrow, 3=page up, 4=page down, 5=ctrl left arrow, 6= ctrl right arrow, and 9=end.

INERROR (STRING, LEN) *utils.obj*

Beep to indicate to the user that an error has occurred and display an error message on line 24.

→	STRING	char	string to be written.
→	LEN	integer*2	length of the string.

MENU (MNAME, MITEMS, MAVAIL, IDEF, IRET) *utils.obj*

This subroutine displays an underlined title followed by a vertical list of possible choices. An option can be selected when it is displayed in reverse video (those that do not change into reverse video cannot be selected). The user moves up and down the menu with the up and down arrow keys and selects the highlighted choice by hitting the end key. The array MAVAIL determines which menu items can be selected.

→ MNAME	char*32	array that contains the menu title and all the selections.
→ MITEMS	integer*2	number of items in the array MNAME including the title.
→ MAVAIL	integer*2	array that determines which items can be selected. A zero disables the item while a one enables it.
→ IDEF	integer*2	the default item number.
← IRET	integer*2	The item selected.

YESNO (I, ROW, COL, IRET) *utils.obj*

Print the default value (Y or N) to the screen at (ROW, COL) and accept only a "Y" or an "N" as input.

↔ I	integer*2	the default on input and the answer on output. 0=yes 1=no.
→ ROW	integer*2	row address of the start of the edit field ($0 \leq ROW \leq 24$).
→ COL	integer*2	column address of the start of the edit field ($0 \leq COL \leq 79$).
← IRET	integer*2	Terminating output code where 0=carriage return, 1=up arrow, 2=down arrow, 3=page up, 4=page down, 5=ctrl left arrow, 6= ctrl right arrow, and 9=end.

WAITKY *util1.obj*

This subroutine prints the message "Hit any key to continue" on line 24 and waits for the user to hit a key whereupon the message is deleted from the screen and control returns to the calling program.

Miscellaneous Routines

This section describes the remaining subroutines. These provide some string manipulation routines (BLANK, CONHEX, STRI, STRX, etc), Fortran equivalents of some Basic routines (POKE, WAIT, VARPT, etc.), DOS date and time interfaces, and other utilities needed when the file routines are used (ADDNUL and SUBNUL).

Quick Reference

ADDNUL (STRING, LEN) — Add a null to the end of string.
BLANK (STRING, LEN) — Fill a string with blanks.
CONHEX (I, D) — Convert an integer into a hex string.
GDATE (YEAR, MONTH, DAY) — Get the date.
GTIME (HOUR, MIN, SEC) — Get the time.
HOWLNG (STRING, NTOTAL, LEN) — How long is the string.
IOUTB (I, J) — Output a byte.
IOUTW (I, J) — Output a word.
IPOKEB (I, JJ) — Put a byte into memory.
IPOKEW (I, JJ) — Put a word into memory.
IVARPT (I, JJ) — Get the address of a variable.

LJUST (STRING, LEN) — Left justify a string.
PHEX (I) — Print an integer in hexadecimal.
PHYSAD (N, JJ) — Calculate the physical address of a variable
PWAIT (IP, N, M) — Wait for a condition on an input port.
RETPRO (IRET, I, J, IMAX, JMAX, PAGE, PAGMAX) — Process a return code.
STRI (I, STRING, LEN) — Convert an integer into a string.
STRX (X, STRING, LEN) — Convert a real number into a string.
SUBNUL (STRING, LEN) — Remove a null from a string.

Subroutine Descriptions

ADDNUL (STRING, LEN) *fsel.obj*

Find the first blank in the string STRING and substitute a null. This subroutine converts a normal Fortran string into an ASCIIZ string.

↔ STRING	char	string in which the null is placed.
→ LEN	integer*2	maximum length of the string.

BLANK (STRING, LEN) *util1.obj*

Fill the string STRING with LEN number of blanks.

← STRING	char	string into which blanks are inserted.
→ LEN	integer*2	number of blanks to insert.

CONHEX (I, D) *util2.obj*

Convert the integer I into a four character hexadecimal string.

→ I	integer*2	integer to convert.
← D	char*4	string to hold hexadecimal conversion.

GDATE (YEAR, MONTH, DAY) *forext.obj*

Get the date.

← YEAR	integer*2	current year (e.g. 1986).
← MONTH	integer*2	month (1-12).
← DAY	integer*2	day (1-31).

GTIME (HOUR, MIN, SEC) *forext.obj*

Get the time.

← HOUR	integer*2	hour (0-23).
← MIN	integer*2	minute (0-59).
← SEC	integer*2	seconds (0-59).

HOWLNG (STRING, NTOTAL, LEN) *utils.obj*

Determine how many non-blank characters are at the beginning of the string.

→	STRING	char	input string.
→	NTOTAL	integer*2	maximum length of the string.
←	LEN	integer*2	actual length of the satring.

IOUTB (I, J) *forext.obj*

Send a byte to an I/O port.

→	I	integer*2	byte to output.
→	J	integer*2	port address.

IOUTW (I, J) *forext.obj*

Send a word to an I/O port.

→	I	integer*2	word to output.
→	J	integer*2	port address.

IPOKEB (I, JJ) *forext.obj*

Poke a byte into memory (similar to the Basic POKE).

→	I	integer*2	byte to put into memory.
→	J	integer*2	memory address where JJ (1)=offset and JJ (2)=segment.

IPOKEW (I, JJ) *forext.obj*

Poke a word into memory .

→	I	integer*2	word to put into memory.
→	J	integer*2	memory address where JJ (1)=offset and JJ (2)=segment.

IVARPT (I, JJ) *forext.obj*

Get the address (segment and offset) of a variable.

→	I	any	variable.
←	JJ	integer*2	memory address where JJ (1)=offset and JJ (2)=segment.

LJUST (STRING, LEN) *util2.obj*

Left justify the string.

↔ STRING	char	on input string to left justify and on output the justified string.
→ LEN	integer*2	maximum length of the string.

PHEX (I) *utils.obj*

Print the integer in hexadecimal format.

→ I	integer*2	integer to print in hex.

PHYSAD (N, JJ) *utils.obj*

Calculate the physical address of the variable N. This routine is used to supply the values to set the DMA page register and address register.

→ N	any	variable.
← JJ	integer*4	the physical address of N (i.e. 00000 to FFFFF).

PWAIT (IP, N, M) *utils.obj*

Wait until the byte read from port IP exclusive OR'ed with M and AND'ed with N is non zero (same as Basic WAIT).

→ IP	integer*2	port address.
→ N	integer*2	AND mask.
→ M	integer*2	exclusive OR mask.

RETPRO (IRET, I, J, IMAX, JMAX, PAGE, PAGMAX) *retpro.obj*

Take the return from an user-friendly input routine and update the row, column, and page pointers. This subroutine facilitates page editing.

IRET=0 carriage return move right or to next line if I=IMAX I=I+1
IRET=1 up arrow move up one line or to bottom if J=1 J=J+1
IRET=2 down arrow move down one line or to top if J=JMAX J=J-1
IRET=3 page up increment page PAGE=PAGE+1
IRET=4 page down decrement page PAGE=PAGE-1
IRET=5 cntrl left arrow move left or to previous line if I=1 I=I-1
IRET=6 cntrl right arrow move right or to next line if I=IMAX I=I+1

→ IRET	integer*2	return from user friendly input routine.
↔ I	integer*2	column field pointer.
↔ J	integer*2	row field pointer.
→ IMAX	integer*2	maximum number of column edit fields.

→ JMAX	integer*2	maximum number of row edit fields.
↔ PAGE	integer*2	current page.
→ PAGMAX	integer*2	maximum number of pages.

STRI (I, STRING, LEN) *util3.obj*

Convert the integer into a string stripping away leading blanks and return its length.

→ I	integer*4	number to be converted.
← STRING	char	string representation of I.
↔ LEN	integer*2	number of characters in string. As input, this is the maximum length of STRING. As output, it is the number of characters in STRING.

STRX (X, STRING, LEN) *util3.obj*

Convert the real number into a string stripping away leading blanks, trailing zeros and decimal point and return its length.

→ I	real*4	number to be converted.
← STRING	char	string representation of X.
↔ LEN	integer*2	number of characters in string. As input, this is the maximum length of STRING. As output, it is the number of characters in STRING.

SUBNUL (STRING, LEN) *fsel.obj*

Find the first null in the string and change it to a space. This routine converts an ASCIIZ string into a normal Fortran string.

↔ STRING	char	string in which the null is to be replaced by a space.
→ LEN	integer*2	maximum length of the string.

APPENDIX D: PROGRAM SOURCE LISTINGS

This appendix contains the source listings for the profilometer software. The source code is contained in files with the MS DOS extension .FOR for Fortran code, and .ASM for assembler code. Table 14 lists all of the profilometer subroutines in alphabetical order and indicates the name of the text file containing the source code for that routine. The remainder of this appendix consists of the listings of these files, arranged in alphabetical order by file name.

Table 14. Directory of source files for the profilometer software.

Routine	*File*	*Description*
ACAL	calib.for	Calibrate an analog data channel.
ADCHECK	adcheck.for	Check the calibration of th A/D and D/A converters.
ADSET	adnew.asm	Set up the data collection parameters and the interrupt routine.
A2DONE	iosubs.for	Collect A/D on channel ICH.
AVEVEL	sigsubs.for	Average and decimate a (speed) signal.
BATCH	batch.for	Process a list of data files.
CALDA	iosubs.for	Set calibration D/A.
CALIB	calib.for	Calibrate the analog hardware and check the height sensors.
CALREL	iosubs.for	Switch calibration relay.
CHKSAT	chksat.for	Check the raw transducer signals for saturation.
CONFIG	config.for	Select which data to collect.
DEBIAS	sigsubs.for	Cubtract bias from signal in real*4 array.
DTCLEAR	iosubs.for	Clear the Data Translation board.
DTCLOCK	iosubs.for	Set the A/D clock on the Data Translation board.
FILCLK	iosubs.for	Set the filter clock
GETELV	getelv.for	Get elevation profiles from tape.
GETLEN	getlen.for	Prompt the user for some type of length measure or range.
GOAHED	process.for	Warn the user that some processing needs to be done.
GRCURS	plotsubs.for	Wait for the user to hit a key, then update plot parameters.
HIPASS	sigsubs.for	Filter a signal with a hi-pass filter.
IAVE	sigsubs.for	Average value of signal in integer*2 array.
INITIO	iosubs.for	Initialize I/O.
INITP	initp.for	Initialize status variables and check the A/D board and the floating point processor.
IOEX	ioex.for	Present a menu of options to exercise the input/output hardware.
LABEL	plotsubs.for	Convert a real number into a string for Halo.
LOADT	loadtape.for	Load and initialize tape
LOGO	logo.for	Draw the logo for the profilometer.
LOPASS	lopass.for	Smooth a signal.
LRSLOP	sigsubs.for	Calculate slope of signal using a linear regression.
MAIN	profmain.for	Show the Logo and offer the main menu to the user.
MEASURE	measure.for	Generate the menu for measuring data.
MINV	minv.for	Matrix inversion.
PLOT	plot.for	Plot data using Halo subroutines.
PLTELV	plotelv.for	Set up plots of profile elevation.
PLTRAW	plotraw.for	Set up plots of raw signals.
PLTRUT	plotrut.for	Set up plots of rut-depth and roughness signals.

Table 14. Directory of source files for the profilometer software (continued).

Routine	*File*	*Description*
PLTSEL	plotsel.for	Prompt user for the selection of channels and plotting ranges.
PRFCMP	prfcmp.for	Convert raw data into slope profile, rut depth, IRI roughness, and elevation profile.
PRFELV	sigsubs.for	Compute compressed elevation profile from slope.
PRFIRI	prfiri.for	Filter a slope profile signal using the IRI quarter-car simulation.
PROCESS	process.for	Generate the menu for viewing data and call the appropriate subroutines.
PRTLF	prtnum.for	Add carriage returns after each line.
PRTNUM	prtnum.for	Print numerics averaged over a specified interval.
PULSE	pulse.for	Check the calibration of the distance sensor.
PULTST	pulsetst.asm	Set up the interrupt and data collection routine for the distance pulser check.
PUTYN	prtnum.for	Put Y or N in specified screen location.
RAVE	sigsubs.for	Average value of signal in real*4 array.
RDSET	rdwrtape.for	Read in SETUP array from a text file.
RDTAPD	rdtapd.for	Read numerical data from processed file.
RDTAPE	rdwrtape.for	Read binary data.
RESTOR	iosubs.for	Restore analog signal conditioning unit.
RUTCMP	rutcmp.for	Compute, average, and decimate a rut-depth signal.
SATMAX	chksat.for	Check raw data signal for saturation at upper limit.
SATMIN	chksat.for	Check raw data signal for saturation at lower limit.
SCLDWN	plotsubs.for	Scale a variable down.
SCLUP	plotsubs.for	Scale a variable up.
SETAD	iosubs.for	Set up the A/D parameters on the Data Translation board.
SETDMA	iosubs.for	Set up the DMA controller.
SETSTM	setstm.for	Calculate coefficients for quarter-car simulation.
SETUPS	setup.for	Edit the transducer information.
STARTAD	startad.for	Start the data collection.
TCHECK	calib.for	Check a height transducer.
TEST	test.for	Collect data.
TIKSET	plotsubs.for	Determine first and last tick marks in a given range.
TSTDIS	tstdis.for	Display summary of test parameters.
TWAIT		Wait for a time interval.
UNLDTP	unloadtp.for	Unload the tape.
UPDSET	rdwrtape.for	Update the SETUP array that begins the current data file.
WRTAPE	rdwrtape.for	Write binary data.
WRTSCR	wrtscr.for	Read names and coordinates from file, create screen display.
WRTSET	rdwrtape.for	Write the SETUP array to a text file.

Table 14. Directory of source files for the profilometer software (continued).

Routine	*File*	*Description*
YESNOL	prtnum.for	Get Yes/No answer and set logical variable.
ZOFF	iosubs.for	Set the offset on an analog card.

adcheck.for

```
$TITLE:'A/D CALIBRATION CHECK'
$STORAGE:2
$NOFLOATCALLS

      SUBROUTINE ADCHECK

$INCLUDE:'IOPARMS'
      REAL NSEZER,NSEREF
      CALL CLRSCR
      CALL SETCUR(1,0)
      WRITE(*,9030)
9030  FORMAT('THIS IS A CHECK OF THE A/D AND D/A CALIBRATION'\)

C     DISABLE D/A I.E. SET CALHI=CALLO
      CALL IOUTB(DADIS,CNTRL)

C     TURN OFF CAL RELAY AND SHUNT CAL RELAY
      CALL CALREL(15,0)
      CALL IOUTB(SHOFF,CNTRL)
      CALL TWAIT(.4)

C     GET A SECOND OF DATA
      CALL A2DONE(7,1,100.,100,AVZERO,NSEZER)
      CALL SETCUR(3,0)
      WRITE(*,9000)AVZERO,NSEZER
9000  FORMAT('A/D ZERO = ',F7.4,'VOLTS  NOISE= ',F7.4,' VRMS'\)

C     NOW CHECK REFERENCE VOLTAGE

C     SELECT REFERENCE VOLTAGE=2.5
      CALL IOUTB(SHON,CNTRL)
      CALL TWAIT(.4)
      CALL A2DONE(7,1,100.,100,AVREF,NSEREF)
      CALL SETCUR(4,0)
      WRITE(*,9010)AVREF,NSEREF
9010  FORMAT('A/D REFERENCE = ',F7.4,'VOLTS  NOISE= ',F7.4,' VRMS'\)

C     CORRECT REFERENCE FOR ZERO SHIFT
      VREF=AVREF-AVZERO
      CALL SETCUR(5,0)
      WRITE(*,9020)VREF
9020  FORMAT('CORRECTED REFERENCE VOLTAGE= ',F7.4\)

C     PRINT OUT PASS OR WARNING MESSAGE
      IF( ABS(VREF-2.5) .GT. 0.015)THEN
      CALL SETCUR(7,0)
      WRITE(*,'(A\)')'***WARNING*** A/D SHOULD BE CALIBRATED'
      ELSE
      CALL SETCUR(7,0)
      WRITE(*,'(A\)')'A/D CALIBRATION IS OK'
      ENDIF

C     CHECK D/A CALIBRATION
C     TURN OFF REFERENCE AND TURN ON D/A
```

```
      CALL SETCUR(9,0)
      WRITE(*,'(A\)')'CHECKING D/A GAIN CALIBRATION'
      CALL CALREL(15,1)
      CALL TWAIT(.4)
      V=2.5
      CALL CALDA(V)
      CALL A2DONE(7,1,100.,100,AVZERO,NSEZER)
      V1=-V
      CALL CALDA(V1)
      CALL SETCUR(10,0)
      WRITE(*,9040)AVZERO,V
9040  FORMAT('MEASURED ',F7.4,'   SHOULD BE= ',F7.4\)
      CALL TWAIT(.4)
      CALL A2DONE(7,1,100.,100,AVREF,NSEREF)
      CALL SETCUR(11,0)
      WRITE(*,9040)AVREF,V1
      VREF=AVZERO-AVREF
      IF( ABS(VREF-5.) .GT. .03)THEN
      CALL SETCUR(13,0)
      WRITE(*,'(A\)')'***WARNING*** D/A SHOULD BE CALIBRATED'
      ELSE
      CALL SETCUR(13,0)
      WRITE(*,'(A\)')'D/A  GAIN CALIBRATION IS OK'
      ENDIF
      CALL WAITKY
      RETURN
      END
```

ADNEW.ASM

```
      TITLE A/D ROUTINES
      PAGE  ,132
DATA  SEGMENT PUBLIC 'DATA'
BUFT  DD    ?                   ;BUFFER TABLE BASE ADDRESS
BUFST DD    ?                   ;BUFFER STATUS TABLE BASE ADDRESS
NUMB  DW    ?                   ;NUMBER OF BUFFERS
BYTB        DW    ?                   ;BYTES PER BUFFER
MAXB  DW    ?                   ; MAX # OF BUFFERS TO FILL
BUFCNT      DD    ?                   ;BUFFER FILLED COUNT ADDRESS
DONEA DD    ?                   ;DONE FLAG ADDRESS
TEMP  DW    ?                   ;TEMPORARY FOR ISR
CURB  DD    ?                   ;CURRENT BUFFER # ADDRESS
DATA  ENDS
DGROUP      GROUP DATA
CODE  SEGMENT     'CODE'
      ASSUME CS:CODE,DS:DGROUP,SS:DGROUP;
DSSAVE      DW    ?
;
;     ADSET(CURB,BUFT,BUFST,NUMB,BYTB,MAXB,BUFCNT,DONE)
;          CURB=A/D CURRENT BUFFER
;          BUFT=INTEGER*4  BUFFER ADDRESS TABLE BUFT(NUMB)
;          BUFST=INTEGER*2  BUFFER STATUS TABLE BUFST(NUMB)
;          NUMB=INTEGER*2 NUMBER OF BUFFERS
;          BYTB=INTEGER*2 NUMBER OF BYTES PER BUFFER
;          MAXB=INTEGER*2 MAXIMUM NUMBER OF BUFFERS TO FILL
;          BUFCNT=INTEGER*2 NUMBER OF BUFFERS FILLED
;          DONE=INT*2 DONE FLAG -1=DONE >1 =ERROR  0=NOT DONE
PUBLIC      ADSET
ADSET PROC  FAR
      PUSH  BP                  ;SAVE BP
      MOV   BP,SP
      MOV   DSSAVE,DS           ;SAVE DS FOR INT
      LES   BX,DWORD PTR [BP+6]       ;GET DONE ADDRESS
      MOV   WORD PTR DONEA,BX ;SAVE OFFSET
      MOV   WORD PTR DONEA+2,ES       ;SAVE SEGMENT
      LES   BX,DWORD PTR [BP+10]      ;GET BUFFER COUNT ADDRESS
      MOV   WORD PTR BUFCNT,BX        ;SAVE OFFSET
      MOV   WORD PTR BUFCNT+2,ES      ;SAVE SEGMENT
      LES   BX,DWORD PTR [BP+14]      ;GET MAX # OF BUFS TO FILL ADDR
      MOV   AX,ES:[BX]          ;GET MAXB
      MOV   MAXB,AX                   ;SAVE IT
      LES   BX,DWORD PTR [BP+18]      ;GET BYTES PER BUFFER ADDR
      MOV   AX,ES:[BX]          ;GET BYTB
      MOV   BYTB,AX                   ;SAVE IT
      LES   BX,DWORD PTR [BP+22]      ;GET NUM # OF BUFFERS ADDR
      MOV   AX,ES:[BX]          ;GET NUMB
      MOV   NUMB,AX                   ;SAVE IT
      LES   BX,DWORD PTR [BP+26]      ;GET STATUS TABLE ADDRESS
      MOV   WORD PTR BUFST,BX ;SAVE OFFSET
      MOV   WORD PTR BUFST+2,ES       ;SAVE SEGMENT
      LES   BX,DWORD PTR [BP+30]      ;GET BUFFER TABLE ADDR
      MOV   WORD PTR BUFT,BX  ;SAVE OFFSET
      MOV   WORD PTR BUFT+2,ES        ;SAVE SEGMENT
      LES   BX,DWORD PTR [BP+34]      ;GET CURRENT BUFFER ADDRESS
```

```
      MOV   WORD PTR CURB,BX  ;SAVE OFFSET
      MOV   WORD PTR CURB+2,ES      ;SAVE SEGMENT
;
;     SET UP INTERRUPT VECTOR
;
      CLI                         ;DISABLE INTS
      PUSH  DS                    ;SAVE DS
      MOV   DX,OFFSET ISR               ;GET VECTOR OFFSET
      PUSH  CS
      POP   DS                    ;DS=SEGMENT FOR INT ROUTINE
      MOV   AL,0AH                      ;INTERRUPT VECTOR #
      MOV   AH,25H                      ;SET VECTOR FUNCTION
      INT   21H                   ;SET IT
      POP   DS                    ;RECOVER DS
;
;     ENABLE IRQ2 ON 8259
;
      IN    AL,21H                      ;GET CURRENT MASKS
      AND   AL,11111011B                ;RESET IRQ2
      OUT   21H,AL
      MOV   SP,BP
      POP   BP                    ;RECOVER BP
      STI                         ;ENABLE INTS
      RET   32                    ;8 ARGS*4 BYTES
ADSET ENDP
;
;     EQUATES FOR ISR
;
PCTRL EQU   307H                  ;8255 CONTROL REG
DTCOM EQU   2EDH                  ;A/D COMMAND REG
DTSTAT      EQU   2EDH                  ;A/D STATUS REG
DTDATA      EQU   2ECH                  ;A/D DATA REG
CWAIT EQU   4                     ;COMMAND WAIR
RWAIT EQU   5                     ;READ WAIT
CDMA  EQU   1EH                   ;A/D DMA COMMAND
CRAD  EQU   0EH                   ;A/D NON-DMA COMMAND
INTD  EQU   310H                  ;INTERRUPT DISABLE ADDRESS
;
;     CLOCK INTERRUPT ROUTINE-POINT DMA TO NEXT BUFFER
;
ISR   PROC  NEAR
      CLI               ;NO INTS
      PUSH  AX                    ;SAVE REGISTERS
      PUSH  BX
      PUSH  CX
      PUSH  DX
      PUSH  DS
      PUSH  ES
      MOV   AX,DSSAVE             ;GET DS
      MOV   DS,AX                 ;SET IT
      MOV   DX,DTSTAT             ;GET STATUS ADDR
      IN    AL,DX                 ;GET STATUS
      TEST  AL,80H                      ;ERROR?
      JE    ISRA
      JMP   DTERR                 ;YES-EXIT
;
;     INDICATE CURRENT BUFFER IS FULL
;
```

```
ISRA: LES    BX,CURB                  ;GET CURB ADDRESS
      MOV    AX,ES:[BX]         ;GET CURRENT BUFFER #
      MOV    CX,AX              ;COPY IT
      SHL    CX,1               ;*2 FOR OFFSET INTO STATUS TABLE
      LES    BX,BUFST           ;GET STATUS TABLE BASE ADDRESS
      ADD    BX,CX              ;ADD OFFSET
      MOV    WORD PTR ES:[BX],0FFFFH ;INDICATE FULL
      MOV    TEMP,BX                  ;SAVE OFFSET FOR LATER
;
;     CHECK FOR DONE
;
      LES    BX,BUFCNT          ;GET BUFFER COUNT ADDRESS
      INC    WORD PTR ES:[BX]   ;INCREMENT BUFFER COUNT
      MOV    CX,MAXB                  ;GET MAX BUFFERS TO FILL
      CMP    CX,ES:[BX]         ;DONE?
      JNE    DNCHK
;
;     SET DONE FLAG
;
      LES    BX,DONEA           ;GET DONE ADDRESS
      MOV    WORD PTR ES:[BX],0FFFFH ;SET DONE
DNCHK:       LES    BX,DONEA
      CMP    WORD PTR ES:[BX],0       ;DONE?
      JE     ISRC               ;NO
ISR4: MOV    DX,INTD                  ;GET INT DIABLE ADDRESS
      OUT    DX,AL              ;DISABLE INTS
      JMP    ISROT
;
;     NOT DONE - GOTO NEXT BUFFER
;
ISRC: CMP    AX,NUMB                  ;LAST BUFFER
      JNE    ISRD               ;NO
      MOV    AX,0               ;YES- CURB=0
      LES    BX,BUFST           ;ES:[BX]=NEXT BUFFER STATUS ADDR
      JMP    ISRE
ISRD: INC    AX                 ;NEXT BUFFER
      MOV    BX,TEMP                  ;RECOVER LAST OFFSET INTO STATUS
      ADD    BX,2               ;NEXT STATUS
ISRE: CMP    WORD PTR ES:[BX],0       ;EMPTY?
      JNE    OVERE              ;NO-ERROR
      LES    BX,CURB                  ;GET CURB ADDRESS
      MOV    ES:[BX],AX         ;SAVE CURRENT BUFFER
;
;     GET BASE ADDRESS AND PAGE ADDRESS FOR DMA
;
      SHL    AX,1
      SHL    AX,1               ;AX=OFFSET INTO TABLE
      LES    BX,BUFT                  ;GET TABLE ADDRESS
      ADD    BX,AX              ;ADD OFFSET
;
;     SET DMA
;
      MOV    AL,45H
      OUT    11,AL              ;SET DMA MODE
      MOV    AL,0
      OUT    12,AL              ;RESET BYTE FLIP FLOP
      MOV    AX,ES:[BX]         ;GET BASE ADDRESS FOR DMA
      MOV    CX,ES:[BX]+2             ;GET PAGE ADDRESS
```

```
      MOV    BX,BYTB                       ;GET NUMBER OF BYTES
      OUT    2,AL                    ;SET LOW BYTE OF BASE
      MOV    AL,AH                   ;AL=HIGH BYTE
      OUT    2,AL                    ;SET HIGH BYTE OF BASE
      MOV    AL,BL                   ;GET LOW BYTE OF CONV
      OUT    3,AL                    ;SET IT
      MOV    AL,BH                   ;GET HIGH BYTE
      OUT    3,AL                    ;SET IT
      MOV    AX,CX                   ;AX=PAGE
      OUT    83H,AL                        ;SET IT
      MOV    AL,1                    ;ENABLE MASK
      OUT    10,AL
ISROT:       MOV    AL,0
      MOV    DX,PCTRL                ;GET INT FF ADDRESS
      OUT    DX,AL                   ;RESET FLIP FLOP
      INC    AL
      OUT    DX,AL                   ;RE-ENABLE IT
;
;     SIGNAL END OF INT TO 8259
;
      MOV    AL,20H
      OUT    20H,AL
;
;     RECOVER REGS AND EXIT
;
      POP    ES
      POP    DS
      POP    DX
      POP    CX
      POP    BX
      POP    AX
      IRET
;
;     OVERUN ERROR
;
OVERE:       MOV    AX,2
      JMP    DTER1
;
;     ERROR-SET DONE >0
;
DTERR:       MOV    AX,1
DTER1:       LES    BX,DONEA             ;GET DONE ADDR
      MOV    WORD PTR ES:[BX],AX         ;SET DONE
      JMP    ISR4
ISR   ENDP
CODE  ENDS
      END
```

batch.for

```
$TITLE:'BATCH FILE PROCESSOR'
$STORAGE:2
$NOFLOATCALLS

      SUBROUTINE BATCH (DR)
************************************************************
*  Process a list of data files to get profile and rut depth.
*
$INCLUDE:'HANDLES'
$INCLUDE:'SETCOM'
      CHARACTER*16 FILES(80),FILE
      CHARACTER*3 EXT
      CHARACTER*1 DR
      CHARACTER*8 NAME
      INTEGER*2 ROW,COL
      CALL CLRSCR

      EXT='DTA'

C     WRITE MESSAGE
      CALL SETCUR(10,0)
      WRITE(*,'(A\)')'BATCH PROCESSING PROGRAM'
      CALL SETCUR(11,0)
      WRITE(*,'(A\)')'DO YOU WANT TO PROCESS MANY FILES? '
      I=1
      CALL GCUR(ROW,COL)
      CALL YESNO(I,ROW,COL,IRET)
      IF(I .EQ. 0)RETURN

C     GET DRIVE
      CALL DRVSEL(DR)


C     GET FILENAME WITH WILD CODES
      NAME='*'
50    CALL CLRSCR
      CALL WRTSCR('BATCHSCR.       ')
      CALL SETCUR(14,50)
      WRITE(*,9010)DR,NAME,EXT
9010  FORMAT(A,':',A8,'.'A3\)
      CALL GETSTRNG(NAME,8,14,52,IRET)
      CALL FNMAKE(DR,NAME,EXT,FILE,0)
      ISEL=80
      CALL FSELALL(FILE,ISEL,FILES)
      IF(ISEL .EQ. 0)RETURN
      IF(ISEL .EQ. 80)THEN
      WRITE(*,'(A\)')'TOO MANY FILES WERE SELCTED.  REENTER THE ',
     1       'THE NAME'
      CALL WAITKY
      GOTO 50
      ENDIF
      CALL CLRSCR
      CALL SETCUR(8,0)
```

```
      WRITE(*,'(I2,A\)') ISEL,' FILES WERE SELECTED'

C     PROCESS FILES

      CALL KCLEAR
      CALL SETCUR (10,5)
      WRITE (*,'(A\)')
     &   '<Hit the "End" key to stop processing after this file.>'

      DO 100 I = 1, ISEL
      CALL SETCUR(18,0)
      WRITE(*,'(A,I2\)') 'NOW PROCESSING FILE #',I
      CALL ADDNUL (FILES (I),16)
      ACCESS = 2
      CALL HOPEN(FILES(I),HANDLE,ACCESS,IER)
      CALL SUBNUL (FILES (I),16)
      CALL HREAD(HANDLE,SET,2048,RBYTES,IER)
      IF (TSTTYP .EQ. 0 .OR. TSTTYP .EQ. 4) THEN
        CALL CHKSAT (HANDLE, 1)
        IF (.NOT. ITSOK) GO TO 80
      END IF
      IF (TSTTYP .EQ. 3) CALL PRFCMP (HANDLE)
80    CALL HCLOSE (HANDLE,IER)
90    J = IGKEY()
      IF (J .EQ. -79) RETURN
      IF (J .NE. 0) GO TO 90
100   CONTINUE

      RETURN
      END
```

CALIB.FOR

```
$STORAGE:2
$NOFLOATCALLS

      SUBROUTINE CALIB

$INCLUDE:'STATCOM'
$INCLUDE:'SETCOM'
      INTEGER*2 ROW,COL
      CHARACTER*6 BPOS
      CALL CLRSCR

C     WRITE OUT INSTRUCTIONS FOR FIRST STEP
      CALL SETCUR(10,0)
      WRITE(*,9100)
9100  FORMAT('PLACE THE CALIBRATION BAR IN THE MIDDLE POSITION'/
     1 ' DO NOT MOVE AROUND IN THE VEHICLE'/
     2 ' HIT ANY KEY TO START THE CALIBRATION')
      WRITE(*,'(A\)')' '
      CALL WAITKY
      CALL FILCLK(100.0)
      CALL WRTSCR('CALDIS.      ')
      CALL SETCUR(0,0)
      WRITE(*,9000)TSTCON
9000  FORMAT('CALIBRATING CHANNELS FOR  ',A32,'CONFIGURATION'\)

C     WITH THE BAR IN THE ZERO POSITION
C     ZERO ALL CHANNELS AND MEASURE AMPLIFIER GAINS

100   ROW=7
      J=ADSTRT
      DO 200 I=1,NCHAN
      CALL ACAL(J,ROW)
      J=J+1
      IF( J .GT. 7)J=0
      ROW=ROW+2
200   CONTINUE
      CALL WAITKY

C     WITH THE BAR IN THE TOP+BOTTOM POSITIONS- CHECK FOR APPOXIMATE
C     TRANSDUCER GAINS
C     DO ONLY THE HEIGHT AND RUT CHANNELS FOR ANY GIVEN CONFIGURATION
      BPOS='TOP'

      DO 350 I=1,2
      CALL CLRSCR
      CALL SETCUR(10,0)
      WRITE(*,9120)BPOS
9120  FORMAT('PLACE THE CALIBRATION BAR IN THE ',A6,' POSITION'/
     1 ' HIT ANY KEY TO CONTINUE THE CALIBRATION')
      WRITE(*,'(A\)')' '
      CALL WAITKY
      CALL WRTSCR('CALDIS2.       ')
      CALL SETCUR(1,41)
      WRITE(*,'(A6\)')BPOS
```

```
      ROW=7
      IF (RPROF) CALL TCHECK(0,ROW,I)
      IF (LPROF) CALL TCHECK(4,ROW,I)
      IF (CRUT) CALL TCHECK(5,ROW,I)
      IF (LRUT) CALL TCHECK(6,ROW,I)
      IF (RRUT) CALL TCHECK(7,ROW,I)
      BPOS='BOTTOM'
      CALL WAITKY
350   CONTINUE


      CALCON=TCONFI
      CALYN=1
      CALL GTIME(IH,IMIN,ISEC)
      CALTIM=IH*3600+IMIN*60+ISEC
      CALL WRTSET
500   RETURN
      END
$PAGE
```

```
C
C     CHECK A TRANSDUCER
C
      SUBROUTINE TCHECK(IC,ROW,IPOS)

$INCLUDE:'SETCOM'
      INTEGER*2 ROW
      REAL HGT(8,2)

      HGT(1,1)=1.036
      HGT(5,1)=1.048
      HGT(6,1)=1.042
      HGT(1,2)=-1.012
      HGT(5,2)=-1.010
      HGT(6,2)=-1.011
      SC=2048./5.

      J=IC+1
C     WRITE NAME
      CALL SETCUR(ROW,1)
      WRITE(*,'(A8\)')CHID(J)

C     WRITE NOMINAL HEIGHT
      CALL SETCUR(ROW,12)
      WRITE(*,'(F6.3\)')HGT(J,IPOS)

C     GET ACTUAL HEIGHT AND COMPUTE ERROR
      CALL A2DONE(IC,1,100.,100,AV,VNSE)
      ZDV=5.*(ZDATA(J)/(GAIN(J)*2048.)-1.)
      HGTA=GAIN(J)*(AV-ZDV)*SC
      ERROR=100.*(HGT(J,IPOS)-HGTA)/HGT(J,IPOS)

C     WRITE ACTUAL HEIGHT
      CALL SETCUR(ROW,25)
      WRITE(*,'(F6.3\)')HGTA

C     WRITE ERROR
      CALL SETCUR(ROW,39)
      WRITE(*,'(F7.3\)')ERROR

C     CHECK FOR WARNING
      CALL SETCUR(ROW,51)
      IF (ABS(ERROR) .LT. 2.)THEN
      WRITE(*,'(A\)')'OK'
      ELSE
      WRITE(*,'(A\)')'WARNING'
      ENDIF
      ROW=ROW+1
      RETURN
      END
$PAGE
```

```
C
C     CALIBRATE ONE ANALOG CHANNEL
C
      SUBROUTINE ACAL(II,ROW)

$INCLUDE:'SETCOM'
      INTEGER*2 ROW,COL
      DIMENSION X(11),Y(11)
      I=II+1
      N=300
      F=300.0
      ION=1
      IOFF=0
      AV=0
      VNSE=0
      CALL SETCUR(ROW,1)
      WRITE(*,9600)II,CHID(I)
9600  FORMAT(I1,3X,A8\)
C     TURN OFF CAL RELAY
      CALL CALREL(II,IOFF)
C     PUT OUT LAST OFFSET
      CALL ZOFF(II,IOFFS(I))
      CALL TWAIT(1.0)
C     GET CURRENT OFFSET
100   CALL A2DONE(II,1,F,N,AV,VNSE)
      IF( ABS(AV) .LT. 4.5)GOTO 200
      IOFFS(I)=0
      CALL ZOFF(II,IOFFS(I))
      CALL INERROR('ADJUST OFFSET POT-HIT ANY KEY',29)
      CALL KCLEAR
150   J=IGKEY()
      IF( J .EQ. 0)GOTO 150
      CALL CLRLIN(24)
      IF (J .EQ. 43) GOTO 800
      GOTO 100
200   CALL SETCUR(ROW,15)
      WRITE(*,'(F7.4\)')AV
      CALL SETCUR(ROW+1,15)
      WRITE(*,'(F7.4\)')VNSE
      IF( ABS(AV) .LT. 0.04)GOTO 300
      AV=-(AV-IOFFS(I)*4.8/128)
      IOFFS(I)=NINT(AV/4.8*128)
250   CALL ZOFF(II,IOFFS(I))
300   CALL TWAIT(.5)
      CALL A2DONE(II,1,F,N,AV,VNSE)
      IF(ABS(AV) .LT. .04)GOTO 400
      IF( AV .LT. 0)IOFFS(I)=IOFFS(I)+1
      IF( AV .GT. 0)IOFFS(I)=IOFFS(I)-1
      IF(ABS(IOFFS(I)) .GT. 127)GOTO 900
      GOTO 250
400   CALL SETCUR(ROW,25)
      WRITE(*,'(F7.4\)')AV
      CALL SETCUR(ROW+1,25)
      WRITE(*,'(F7.4\)')VNSE

      ZDATA(I)=AV
      CALL CLRLIN(24)
```

```
      CALL SETCUR(24,30)
      WRITE(*,'(A\)')'MEASURING AMPLIFIER GAIN'
C
C     NOW DO GAIN
C     TURN ON CAL RELAY
      CALL CALREL(II,ION)
      CALL ZOFF(II,0)
      CALL TWAIT(.2)
      XAV=0.0
      YAV=0.0
      VMAX=.9*(5.0/AMPGN(I))
      IF(VMAX .GT. 4.8)VMAX=4.8
      V=VMAX/5.
      K=NINT(V/5.0*2048)
      V=K*5.0/2048
      DO 500 K=-5,5
      V1=V*K
      CALL CALDA(V1)
      CALL TWAIT(.5)
      CALL A2DONE(II,1,F,N,AV,VNSE)
      X(K+6)=V1
      Y(K+6)=AV
      XAV=XAV+V1
      YAV=YAV+AV
500   CONTINUE
      XAV=XAV/11.0
      YAV=YAV/11.0
      A=0.0
      B=0.0
      DO 600 K=1,11
      A=A+(X(K)-XAV)*(Y(K)-YAV)
      B=B+(X(K)-XAV)**2
600   CONTINUE
      GAIN(I)=A/B
      AMPGA(I)=GAIN(I)
C
C     RESTORE CARD TO ORIGINAL STATUS
      CALL CALREL(II,IOFF)
      CALL ZOFF(II,IOFFS(I))
      CALL CALDA(0.0)
      CALL IOUTB(4,#307)
      CALL CLRLIN(24)
      CALL SETCUR(ROW,35)
      WRITE(*,9200)AMPGN(I),AMPGA(I)
9200  FORMAT(F9.4,3X,F9.4\)
C
C     CALCULATE FULL SCALE
      GAIN(I)=XDUCGN(I)/GAIN(I)*5
      CALL SETCUR(ROW,57)
      WRITE(*,9300)GAIN(I),UNITS(I)
9300  FORMAT(F9.4,2X,A8\)
      GAIN(I)=GAIN(I)/2048.
      ZDATA(I)=ZDATA(I)*GAIN(I)*2048./5.0+2048.*GAIN(I)
800   RETURN
900   CALL INERROR('UNABLE TO ADJUST OFFSET',23)
      CALL CALREL(II,IOFF)
      IOFFS(I)=0
      CALL ZOFF(II,IOFFS(I))
```

```
RETURN
END
```

CHKSAT.FOR

```
$TITLE:'SUBROUTINE CHKSAT'
$STORAGE:2
$NOFLOATCALLS

      SUBROUTINE CHKSAT (HANDLE, AUTO)
**********************************************************************
*  This subroutine checks the raw transducer signals for saturation.
*
*  --> HANDLE int*2  handle for tape file that gets checked.
*  --> AUTO   int*2  code indicating interactive or auto modes.
*                    0 = interactive, 1 = don't truncate if error,
*                    2 = truncate if error, 3 = interactive if err.
**********************************************************************

$INCLUDE:'BUFCOM'
$INCLUDE:'HANDLES'
$INCLUDE:'SETCOM'
**********************************************************************
      INTEGER*2 IERR,MIN (8), MAX (8), AUTO
      INTEGER*4 I, NS,LMIN (8), LMAX (8), NMIN (8), J,
     &          NMAX (8), OFF, IB, NBUF, COUNT (8), LSAT1

*  Set bogus parameters for bounce test.

      IF (TSTTYP .EQ. 1 .AND. NSAMP .GT. 4110) NSAMP = 4112
*
*  Set the number of samples contained in the PC buffer and the
*  number of buffers.
*
      MAXBUF = MXBFSZ
      NS = MAXBUF * 2 / NCHRAW
      IF (NS .GT. NSAMP) NS = NSAMP
      NBUF = NSAMP / NS
      IF (MOD (NSAMP, NS) .NE. 0) NBUF = NBUF + 1
*
*  Loop to read and check data, a buffer at a time.
*
      DO 30 IB = 0, NBUF - 1
      CALL SETCUR (20,0)
      WRITE (*,9500)IB+1,NBUF
9500  FORMAT('CHECKING RAW DATA. LOOKING AT BUFFER #',I3,' OF',I3\)
      CALL SETCUR(21,0)
      WRITE (*,'(''READING...     ''\)')
        OFFSET = IB * NS * NCHRAW * 2
        IF (IB .EQ. NBUF - 1) NS = NSAMP - IB * NS
        BYTES = NS * NCHRAW * 2
        CALL RDTAPE (HANDLE, PCBUFI, OFFSET, BYTES, IERR)
      CALL SETCUR(21,0)
      WRITE(*,'(''CHECKING...    ''\)')
*
*  Initialize variables for searching.
*
        IF (IB .EQ. 0) THEN
          DO 10 I = 1, NCHRAW
```

```
          IF (I .EQ. ICHV) THEN
            IF (TSTTYP .EQ. 1) THEN
              DO 5 J = I, NS * NCHRAW + I, NCHRAW
    5         PCBUFI (J) = 3800
              OFFSET = IB * NS * NCHRAW * 2
              BYTES = NS * NCHRAW * 2
              CALL WRTAPE (HANDLE, PCBUFI, OFFSET, BYTES, IERR)
            END IF
            MIN (I) = PCBUFI (I)
            MAX (I) = PCBUFI (I)
          ELSE IF (I .EQ. ICHA1 .OR. I .EQ. ICHA2) THEN
            MIN (I) = 10
            MAX (I) = 4090
          ELSE
            MIN (I) = 400
            MAX (I) = 3700
          END IF

          COUNT (I) = 0
          NMIN (I) = 0
          LMIN (I) = 0
          NMAX (I) = 0
          LMAX (I) = 0
   10     CONTINUE
        END IF
*
*   Check all channels in this buffer.
*
        OFF = OFFSET / NCHRAW / 2
        DO 20 I =1, NCHRAW
          CALL SATMAX (PCBUFI (I), NCHRAW, NS, OFF, MAX (I),
     &                 COUNT (I), NMAX (I), LMAX (I))
          CALL SATMIN (PCBUFI (I), NCHRAW, NS, OFF, MIN (I),
     &                 COUNT (I), NMIN (I), LMIN (I))
   20   CONTINUE
   30 CONTINUE
*
*   Merge Max, Min saturations.
*
      DO 40 I =1, NCHRAW
        NSAT (I) = NMAX (I) + NMIN (I)
        LSAT (I) = LMIN (I)
        IF (LMAX (I) .GT. LMIN (I))LSAT (I) = LMAX (I)
   40 CONTINUE

*   Convert min and max speed values to eng. units.

      ITSOK = .TRUE.
      IF (ICHV .NE. 0) THEN
        VELMIN = MIN (ICHV) * GAIN (3) - ZDATA (3)
        VELMAX = MAX (ICHV) * GAIN (3) - ZDATA (3)
      THEMAX = 4090. * GAIN (3) - ZDATA (3)
      IF (VELMIN * SCLFV .GT. 3. .AND. VELMAX .LT. THEMAX)
     &                   NSAT (ICHV) = 0
      END IF

*   Set logical variable if any saturation was found
```

```
      LSAT1 = PASSA
      DO 210 I = 1, NCHRAW
        IF (NSAT (I) .NE. 0)THEN
          ITSOK = .FALSE.
          IF (LSAT (I) .LT. LSAT1) LSAT1 = LSAT (I)
        END IF
  210 CONTINUE
      IF (TSTTYP .EQ. 0 .OR. TSTTYP .EQ. 4) TSTTYP = 3
      IF (TSTTYP .EQ. 1) TSTTYP = 5

*  Guard against low or negative speed.

      IF (ICHV .NE. 0) THEN
        IF (VELMIN * SCLFV .LT. 3.) THEN
          TSTTYP = 4
          ITSOK = .FALSE.
        END IF
      END IF

*  Display results if interactive and it's ok.

      IF ((AUTO .EQ. 0) .AND. ITSOK) THEN
        CALL TSTDIS
        CALL WAITKY

*  Display results and prompt user if it's not ok.

      ELSE IF((AUTO .EQ. 3 .OR. AUTO .EQ. 0) .AND. .NOT. ITSOK) THEN
        CALL TSTDIS
        CALL SETCURS (23,0)
        WRITE (*,9000)
9000    FORMAT ('Do you want to shorten the run to eliminate ',
     &          'questionable data?'\)
        I = 0
        CALL YESNO (I, 23, 64, IRET)
        IF (I .EQ. 1) THEN
          CALL CLRLIN (23)
          CALL SETCURS (23,0)
          WRITE (*,'(A\)') 'End run at X = '
          XUL = PASSA * DELTAX
          XLL = 0
          X = (LSAT1 - 1) * DELTAX
          CALL GETR (X, XLL, XUL, 23, 16, 9, '(F9.2\)', IRET)
          PASSA = X / DELTAX - 1
          IF (LSAT1 .GT. PASSA) THEN
          DO 50 I = 1, NCHRAW
   50     NSAT(I) = 0
        END IF
        END IF

*  If it's not ok, and AUTO=2, and the speed was ok, then fix it.

      ELSE IF (AUTO .EQ. 2 .AND. .NOT. ITSOK) THEN
        IF (TSTTYP .EQ. 3) PASSA = LSAT1 - 1
      END IF

*  Set defaults for viewing data.
```

```
      AVEBAS = 50.
      FLTBAS = 50.
      XCURS = 5.
      XRANGE = 100.
      PSTART = 0.
      PSTOP = PASSA * DELTAX
      PINC = 100.

      CALL UPDSET (HANDLE)
      RETURN
      END
$PAGE
```

```
***********************************************************************
      SUBROUTINE SATMAX (ARRAY, NCH, NS, OFFSET, MAX, COUNT, NSAT,
     &                   LSAT)
***********************************************************************
*  This checks a raw data signal for saturation at an upper limit.
*  It finds the max value of the signal, and looks for 2 or more
*  consecutive samples at that limit.
*
*  --> ARRAY  int*2  2-D input array.  Channel 1 is checked.
*  --> NCH    int*4  number of channels in ARRAY.
*  --> NS     int*4  number of samples in ARRAY.
*  --> OFFSET int*4  samples previously processed.
*  <-> MAX    int*2  maximum value in signal.  Initially, MAX
*                    should be given a valid value.
*  <-> COUNT  int*4  counter to see if signal stays at max
*                    level for 2 adjacent samples.
*  <-> NSAT   int*4  number of saturations in signal.
*  <-> LSAT   int*4  location (sample no.) of first saturation.
***********************************************************************
$LARGE: ARRAY

      INTEGER*2 ARRAY(*), MAX, Y
      INTEGER*4 NSAT, LSAT, NFW, I, OFFSET, NS, NCH, COUNT
*
*  Comments? "The source code is obvious."
*
      NFW = NCH * NS
      DO 10 I = 1, NFW, NCH
        Y = ARRAY (I)
        IF (Y .LT. MAX) THEN
          COUNT = 0
        ELSE IF (Y .GT. MAX) THEN
          MAX = Y
          NSAT = 0
          LSAT = 0
          COUNT = 1
        ELSE
          COUNT = COUNT + 1
          IF (COUNT .EQ. 2) THEN
            NSAT = NSAT + 1
            IF (LSAT .LT. 1) LSAT = OFFSET + I / NCH - 1
          END IF
        END IF
   10 CONTINUE
      RETURN
      END
$PAGE
```

```
************************************************************************
      SUBROUTINE SATMIN (ARRAY, NCH, NS, OFFSET, MIN, COUNT, NSAT,
     &                   LSAT)
************************************************************************
*  This checks a raw data signal for saturation at a lower limit.
*  It finds the MIN value of the signal, and looks for 2 or more
*  consecutive samples at that limit.
*
*  --> ARRAY  int*2  2-D input array.  Channel 1 is checked.
*  --> NCH    int*4  number of channels in ARRAY.
*  --> NS     int*4  number of samples in ARRAY.
*  --> OFFSET int*4  samples previously processed.
*  <-> MIN    int*2  minimum value in signal.  Initially, MIN
*                    should be given a valid value.
*  <-> COUNT  int*4  counter to see if signal stays at min
*                    level for 2 adjacent samples.
*  <-> NSAT   int*4  number of saturations in signal.
*  <-> LSAT   int*4  location (sample no.) of first saturation.
************************************************************************
$LARGE: ARRAY

      INTEGER*2 ARRAY(*), MIN, Y
      INTEGER*4 NSAT, LSAT, NFW, I, OFFSET, NCH, NS, COUNT
*
*  Comments? "The source code is obvious."
*
      NFW = NCH * NS
      DO 10 I = 1, NFW, NCH
        Y = ARRAY (I)
        IF (Y .GT. MIN) THEN
          COUNT = 0
        ELSE IF (Y .LT. MIN) THEN
          MIN = Y
          NSAT = 0
          LSAT = 0
          COUNT = 1
        ELSE
          COUNT = COUNT + 1
          IF (COUNT .EQ. 2) THEN
            NSAT = NSAT + 1
            IF (LSAT .LT. 1) LSAT = OFFSET + I / NCH - 1
          END IF
        END IF
   10 CONTINUE
      RETURN
      END
```

```
C
C       CONFIGURE SYSTEM
C
```

CONFIG.FOR

```
$TITLE:'CONFIGURE'
$STORAGE:2
$NOFLOATCALLS

      SUBROUTINE CONFIGURE

$INCLUDE:'STATCOM'
$INCLUDE:'SETCOM'
      CHARACTER*32 IMENU(11)
      INTEGER*2 MA(11)
      MI=11
      DO 10 I=1,5
10    MA(I)=1
      DO 12 I=6,11
12    MA(I)=0

      IMENU(1)='SELECT CONFIGURATION'
      IMENU(2)='LEFT PROFILE'
      IMENU(3)='RIGHT PROFILE'
      IMENU(4)='LEFT AND RIGHT PROFILE'
      IMENU(5)='LEFT + RIGHT PROFILE + MID RUT'
      IMENU(6)='LEFT PROFILE AND LEFT RUT'
      IMENU(7)='LEFT+RIGHT PROFILE+LEFT+MID RUT'
      IMENU(8)='ALL THREE RUTS'
      IMENU(9)='LEFT PROFILE AND ALL RUTS'
      IMENU(10)='RIGHT PROFILE AND ALL RUTS'
      IMENU(11)='LEFT+RIGHT PROFILE + ALL RUTS'

C     SET DEFAULT TO
      IDEF=TCONFI

C     GET SELECTION
50    CALL MENU(IMENU,MI,MA,IDEF,IRET)
      CALL CLRSCR

C     SET CONFIGURATION NUMBER AND STRING
      TCONFI=IRET
      TSTCON=IMENU(IRET+1)

C     READ IN NUMBER OF CHANNELS, A/D START CHANNEL,
C     A/D STOP CHANNEL, BUFFER OFFSETS,AND OTHER VARIABLES
C     FOR PROCESSING

      OPEN(9,FILE='CONFIG.SET ',ACCESS='DIRECT',FORM='FORMATTED',
     1  RECL=54)
      READ(9,1000,REC=TCONFI)(SET(I),I=493,511),
     1      LPROF,RPROF,LRUT,CRUT,RRUT,NCHRAW,NCHPRF,NCHRUT

1000  FORMAT(19(I5),5(L5),3(I5))
      CLOSE(9)
```

```
C     RETURN TO MAIN PROGRAM
      RETURN
      END
```

CNTTST.BAS

```
3 REM 3/12/85 9:40
5 REM CNTTST.BAS
6 REM SETS UP 9513 CHIP AND RELATED CIRCUITRY
7 REM FOR TROUBLE SHOOTING AND CHECK OUT
8 REM
10 PORTA%=&H304
20 PORTB%=&H305
30 PORTC%=&H306
40 CNTRL%=&H307
50 TIMERD%=&H308
60 TIMERC%=&H309
70 INTD%=&H310
80 KEYBD%=&H300
90 INTE%=&H30C
100 CONTROL%=&H90
110 OUT CNTRL%,CONTROL%
120 OUT CNTRL%,2
150 REM RESET COUNTER AND SET UP
160 OUT TIMERC%,&HFF 'RESET
170 OUT TIMERC%,&H5F 'LOAD ALL =0
180 OUT TIMERC%,&HDF 'DISARM ALL
190 OUT TIMERC%,&HE8 'DISABLE SEQUENCING
200 OUT TIMERC%,&H17 'POINT TO MASTER MODE
210 OUT TIMERD%,&HD0 'SEND LOW BYTE
220 OUT TIMERD%,&H49 'SEND HIGH BYTE
230 REM SET UP COUNTER #1 TO COUNT F1 REPEATEDLY (300.02 HZ)
240 OUT TIMERC%,&H1 'POINT TO COUNTER 1 MODE REG
250 OUT TIMERD%,&H21 'SET MODE
260 OUT TIMERD%,&HB
270 OUT TIMERC%,&H9 'POINT TO LOAD REG
280 OUT TIMERD%,&H12
290 OUT TIMERD%,&H1F
300 REM SET UP COUNTER #2 FOR 25.3868 KHZ FOR A/D CLOCK
340 OUT TIMERC%,&H2 'POINT TO COUNTER 2 MODE REG
350 OUT TIMERD%,&H22 'SET MODE
360 OUT TIMERD%,&HB
370 OUT TIMERC%,&HA 'POINT TO LOAD REG
380 OUT TIMERD%,47
390 OUT TIMERD%,&H0
400 REM SET UP COUNTER 3 TO COUNT OUT2 BY 5
440 OUT TIMERC%,&H3 'POINT TO COUNTER 3 MODE REG
450 OUT TIMERD%,&HA5 'SET MODE
460 OUT TIMERD%,&HD3
470 OUT TIMERC%,&HB 'POINT TO LOAD REG
480 OUT TIMERD%,&H5
490 OUT TIMERD%,&H0
540 OUT TIMERC%,&H4 'POINT TO COUNTER 4 MODE REG
550 OUT TIMERD%,&H21 'SET MODE
560 OUT TIMERD%,&H14
570 OUT TIMERC%,&HC 'POINT TO LOAD REG
580 OUT TIMERD%,&HA
```

```
590 OUT TIMERD%,&H0
600 REM SET UP COUNTER 5 FOR FILTER CLOCK
640 OUT TIMERC%,&H5 'POINT TO COUNTER 5 MODE REG
650 OUT TIMERD%,&H22 'SET MODE
660 OUT TIMERD%,&HB
670 OUT TIMERC%,&HD 'POINT TO LOAD REG
680 OUT TIMERD%,119
690 OUT TIMERD%,&H0
700 OUT TIMERC%,&H7E   'LOAD AND ARM ALL
710 OUT TIMERC%,&H61
720 OUT CNTRL%,3
```

GETELV.FOR

```
$TITLE:'SUBROUTINE GETELV'
$NOFLOATCALLS
$STORAGE:2

      SUBROUTINE GETELV (SKPLOT, NSMP, MOVAV1, MOVAV2, QNDPLT, HANDLE,
     &                   IERR)
***********************************************************************
*  This subroutine is for getting elevation profiles from tape so
*  they can be plotted.
*
*    SKPLOT int*4   number of samples to skip before plotting.
*                   This number should be calulated as X/DX.
*    NSMP   int*4   number of samples to plot.
*    MOVAV1 int*4   number of samples in moving average.
*    MOVAV2 int*4   number of samples in 1/2 moving average.
*    QNDPLT logical switch for Quick-n-dirty plotting.
*    HANDLE int*2   handle for file with processed profile.
*    IERR   int*2   error code.  0=cool.
*
*  Important variables unique to this subroutine:
*    START  int*4   samples to skip before reading from tape.
*    N1-N5  int*4   number of samples in five regions.
*    NS     int*4   number of samples to read from tape.
*    NTOT   int*4   total number of samples on tape.
*    SKPELV int*4   no. of samples to skip to get starting elevation.
*    WHICH  int*2   1=slope profile, 3=elevation profile.
***********************************************************************
$INCLUDE:'SETCOM'
$INCLUDE:'BUFCOM'
***********************************************************************
      INTEGER*2 WHICH, IERR, HANDLE
      INTEGER*4 SKPLOT, NSMP, MOVAV1, MOVAV2, START, N1, N2, N3, N4,
     &          N5, NS, NTOT, I, ICH, SKPELV, I1, I2
      LOGICAL QNDPLT
*
*  Calculate sizes of 5 regions, N1 - N5.
*
      N3 = NSMP
      IF (QNDPLT) THEN
        WHICH = 3
        NTOT = NSRTOT
      ELSE
        WHICH = 1
        NTOT = NSPTOT
      END IF
*
      IF (SKPLOT .GE. MOVAV2) THEN
        N2 = MOVAV2
        N1 = 0
        START = SKPLOT - N2
      ELSE
        N2 = SKPLOT
        N1 = MOVAV2 - N2
        START = 0
```

```
      END IF
*
      IF (SKPLOT + N3 + MOVAV1 - MOVAV2 .LE. NTOT + 1) THEN
        N4 = MOVAV1 - MOVAV2
        N5 = 0
      ELSE
        N4 = NTOT + 1 - SKPLOT - N3
        N5 = MOVAV1 - MOVAV2 - N4
      END IF
*
*  If Q-n-D, read elevation data.  Check to see if last point
*  (defined as having zero elevation but not contained in the
*  file) should be added.
*
      IF (QNDPLT) THEN
*     WRITE (6,*) 'IN GETELV, QND. HANDLE, N1,N2,N3,N4,N5,ISTART=',
*     &     HANDLE,N1,N2,N3,N4,N5,START
        NS = N2 + N3 + N4
        CALL RDTAPD (HANDLE, PCBUFR (N1 * NCHPRF + 1), WHICH,
     &              START, NS, IERR)
*     WRITE (6,*) 'IN GETELV, AFTER RDTAPD: NS, IERR=', NS, IERR
        IF (NS .LT. N2 + N3 + N4) THEN
          DO 10 ICH = 1, NCHPRF
   10     PCBUFR ((N1 + NS) * NCHPRF + ICH) = 0
        END IF
*
*  If not Q-n-D, read slope profile data.  Increase NS if
*  needed to get to the next elevation benchmark.  Then integrate
*  slope profile backwards to get elevation.
*
      ELSE
*     WRITE (6,*) 'IN GETELV, NO QND. HANDLE, N1,N2,N3,N4,N5,ISTART=',
*    &      HANDLE,N1,N2,N3,N4,N5,START
        NS = N2 + N3 + N4
        SKPELV = (START + NS) / TRIM
        IF (START + NS .EQ. TRIM * SKPELV) SKPELV = SKPELV - 1
        NS = (SKPELV + 1) * TRIM - 1 - START
        CALL RDTAPD (HANDLE, PCBUFR ((1 + N1) * NCHPRF + 1), WHICH,
     &              START, NS, IERR)
*     WRITE (6,*) 'IN GETELV, AFTER RDTAPD: NS, IERR=', NS, IERR
        IF (SKPELV .EQ. NSRTOT) THEN
          DO 20 ICH = 1, NCHPRF
   20     PCBUFR ((N1 + NS) * NCHPRF + ICH) = 0
        ELSE
          WHICH = 3
          I = 1
          CALL RDTAPD (HANDLE, PCBUFR ((1 + N1 + NS) * NCHPRF + 1),
     &                WHICH, SKPELV, I, IERR)
*     WRITE (6,*) 'AFTER RDTAPD TO GET ELV REF: NS, IERR=', I, IERR
        END IF
        DO 40 ICH = 1, NCHPRF
          I1 = N1 * NCHPRF + ICH
          I2 = (N1 + NS) * NCHPRF + ICH
          DO 30 I =I2, I1,-NCHPRF
   30     PCBUFR (I) = PCBUFR (I + NCHPRF) * COFINT + DELTAX *
     &                PCBUFR (I)
   40   CONTINUE
      END IF
```

```
*
*  Filter profile(s) using the HIPASS subroutine (moving average).
*
      IF (TSTTYP .EQ. 6) RETURN

      DO 50 ICH = 1, NCHPRF
   50 CALL HIPASS (PCBUFR (ICH), NCHPRF, N1, N2 ,N3, N4, N5,
     &             MOVAV1, MOVAV2)
      RETURN
      END
```

GETLEN.FOR

```
$TITLE:'SUBROUTINE GETLEN'
$NOFLOATCALLS
$STORAGE:2

      SUBROUTINE GETLEN (X, XLL, XUL, UNITS, TITLE, PROMPT, IRET)
*********************************************************************
*  This subroutine prompts the user for some type of length measure.
*  or range.  It is used to get plot scales, baselengths, and so
*  forth.  The menu provided the user will have XLL as the first
*  option, XUL as the last, and will include X in the middle.
*
*  <-> X      real*4  number that is updated by the subroutine.
*  --> XLL    real*4  lower limit of allowable values for X.
*  --> XUL    real*4  upper limit of allowable values for X.
*  --> UNITS  char*8  name of units used for X.
*  --> TITLE  char*32 heading for menu used to get X from user.
*  --> PROMPT char*60 prompt to use for "custom". entry.
*  <-> IRET   int*2   return code. 0=ok, 1=cancel. If IRET is initially
*                     < 0, then -1 might be returned, indicating help
*                     was requested.

      INTEGER*2 MA(25)
      CHARACTER*8 UNITS, STR1
      CHARACTER*32 IM (26), TITLE, IM1(25)
      CHARACTER*60 PROMPT
      REAL STNDRD (24), VALUES (24)
      EQUIVALENCE (IM(2),IM1)

      DATA MA/25*1/
      DATA STNDRD /.001,.002,.005,.01,.02,.05, .1,.2,.5,1.,2.,5.,
     &              10.,20.,50.,100.,200.,500.,1000.,
     &              2000.,5000.,10000.,20000., 100000./

*  Build the array VALUES with the menu options for X.  Start with
*  values XLL and XUL and all standard numbers in between.

      IF (X .LT. XLL) X = XLL
      IF (X .GT. XUL) X = XUL

      DO 1 I = 1, 23
        IF (STNDRD (I) .LE. XLL * 1.0001) I1 = I + 1
        IF (STNDRD (I) .LE. XUL * 1.0001) I2 = I + 1
1     CONTINUE

      VALUES (1) = XLL
      DO 2 I = I1, I2 -1
2     VALUES (I + 2 - I1) = STNDRD (I)

      IF (STNDRD (I2 -1) .LT. XUL) THEN
        I2 = I2 + 1
        VALUES (I2 + 1 - I1) = XUL
      END IF

      NLIST = I2 - I1 + 1
```

```
      NMENU = NLIST

*  Now make room for X if it isn't already in the list.

      DO 5 I = 2, NLIST
        IF (X .GT. VALUES (I-1) .AND. X .LT. VALUES (I)) THEN
          NMENU = NMENU + 1
          DO 4 J = NMENU, I + 1, -1
4         VALUES (J) = VALUES (J -1)
          VALUES (I) = X
          GO TO 6
        END IF
5     CONTINUE
6     CONTINUE

*  Create list for MENU subroutine and set default.

      IDEF = 1
      DO 11 I = 1, NMENU
        L = 8
        CALL STRX (VALUES (I), STR1, L)
        IM1 (I) = ' '
        IM1 (I) (9-L:) = STR1
        IM1 (I) (10:) = UNITS
        IF (VALUES (I) .EQ. X) IDEF = I
   11 CONTINUE
      NMENU = NMENU + 3
      IM (1) = TITLE
      IM (NMENU-1) = '      CUSTOM'
      IM (NMENU) =   '      CANCEL'

C   Let user make choice.

   20 CALL MENU (IM, NMENU, MA, IDEF, I)

      IRET = 0
      IF (I .EQ. NMENU - 1) THEN
        IRET = 1
      ELSE IF (I .EQ. NMENU - 2) THEN
        CALL SETCUR (NMENU + 5, 1)
        WRITE (*, '(A\)') PROMPT
        CALL HOWLNG (PROMPT, 60, L)
        CALL GETR (X, XLL, XUL, NMENU + 5, L + 2, 9, '(F9.3\)',I2)
      ELSE
        X = VALUES (I)
      END IF
      RETURN
      END
```

INITP.FOR

```
$STORAGE:2
$NOFLOATCALLS

      SUBROUTINE INITP

$INCLUDE:'IOPARMS'
$INCLUDE:'SETCOM'
$INCLUDE:'STATCOM'
      DOUBLE PRECISION  D1,D2,D3,S1,S2,S3,T1,T2,T3,T4,XE,XDIV,XSUM
      CHARACTER*16 FN
      CHARACTER*1 DR
      CHARACTER*3 EXT
      LOGICAL*2 EXIST
      FN='C:SETUP.SET  '
      CALL CLRSCR
C     INITIALIZE STATUS VARIABLES
      TINIT=0
      CALYN=0
      CALCON=0
      CALTIM=1500

C     READ IN SETUP
      CALL RDSET

C     CHECK DATA TRANSLATION BOARD

C     WRITE MESSAGE
      CALL SETCUR(0,0)
      WRITE(*,'(A\)')'CHECKING DATA TRANSLATION BOARD-'

C     STOP AND CLEAR DT BOARD
      CALL DTCLEAR
C     WAIT FOR STATUS BIT
      CALL PWAIT(DTSTAT,CWAIT,0)
C     SEND TEST COMMAND
      CALL IOUTB(CTST,DTCOM)

C     CHECK TO SEE IF DATA OUT REGISTER INCREMENTS

      IE=0
      DO 10 J=1,255
      CALL PWAIT(DTSTAT,RWAIT,0)
      IF( J .NE. INPB(DTDATA))IE=IE+1
10    CONTINUE
      CALL SETCUR(0,33)
      IF(IE .EQ. 0)THEN
      WRITE(*,'(A\)')'PASSED'
      ELSE
      WRITE(*,9000)IE
9000  FORMAT('FAILED',I3,' TIMES'\)
      ENDIF

C     CHECK FLOATING POINT PROCESSOR
```

```
      CALL SETCUR(1,0)
      WRITE(*,'(A\)')'CHECKING FLOATING POINT PROCESSOR-'
C     SET CONSTANTS
      D1=1.0D0
      D2=10.0D0
      D3=9.99D0**20.0D0
      S1=2.302585092994046D0
      S2=2.718281828459045D0
      S3=9.80188864829535D+19
      T1=.8414709848078965D0
      T2=.5403023058681398D0
      T3=1.557407724654902D0
      T4=.7853981633974483D0
      XE=.000000000000001D0
      IE=0
      XSUM =0
      XDIV=D1/7.0D0
      DO 30 I=1,7
30    XSUM=XSUM+XDIV
      IF(ABS(XSUM-D1) .GT. XE)IE=IE+1
      IF(ABS(DSIN(D1)-T1) .GT. XE)IE=IE+1
      IF(ABS(DCOS(D1)-T2) .GT. XE)IE=IE+1
      IF(ABS(DTAN(D1)-T3) .GT. XE)IE=IE+1
      IF(ABS(DATAN(D1)-T4) .GT. XE)IE=IE+1
      IF(ABS(LOG(D2)-S1) .GT. XE)IE=IE+1
      IF(ABS(EXP(D1)-S2) .GT. XE)IE=IE+1
      IF (ABS(D3-S3) .GT. XE)IE=IE+1
      CALL SETCUR(1,35)
      IF( IE .EQ. 0)THEN
      WRITE(*,'(A\)')'PASSED'
      ELSE
      WRITE(*,9010)IE
9010  FORMAT('FAILED',I2,'TIMES'\)
      ENDIF
      CALL WAITKY
      RETURN
      END
```

IOEX.FOR

```
$STORAGE:2
$NOFLOATCALLS

      SUBROUTINE IOEX

$INCLUDE:'IOPARMS'
      CHARACTER*32 IMENU(12)
        INTEGER*2 MA(12)
      MI=12
        DO 10 I=1,MI
10      MA(I)=1

      IMENU(1)='INPUT/OUTPUT EXERCISER'
      IMENU(2)='SET CALIBRATION D/A'
      IMENU(3)='CALIBRATION RELAY'
      IMENU(4)='SET OFFSET'
      IMENU(5)='READ A/D'
      IMENU(6)='WAIT FOR A SPECIFIED TIME'
      IMENU(7)='CLEAR DATA TRANSLATION BOARD'
      IMENU(8)='SET DATA TRANSLATION CLOCK'
      IMENU(9)='SET FILTER CLOCK'
      IMENU(10)='RESTORE ANALOG'
      IMENU(11)='A/D REFERENCE'
      IMENU(12)='EXIT TO MAIN MENU'

      IDEF=1

C     SKIP OVER WAIT KEY
        GOTO 110
100   CALL WAITKY

C     GET MENU SELECTION

110     CALL MENU(IMENU,MI,MA,IDEF,IRET)
      IDEF=IRET
        CALL CLRSCR
        CALL SETCUR(1,0)
        WRITE(*,'(A\)')IMENU(IRET+1)
      GOTO (500,1000,1500,2000,2500,3000,3500,4000,4500,5000,5500)IRET

C     SET CALIBRATION D/A

500     V=0.0
        CALL SETCUR(12,30)
510   WRITE(*,'(A\)')' VOLTAGE='
        CALL GCUR(IROW,ICOL)
      CALL GETR(V,-5.0,5.0,IROW,ICOL,5,'(F5.2\)',IRET)
      CALL CALDA(V)
      GOTO 100

C     TURN ON/OFF CALIBRATION RELAY

1000    I=0
        CALL SETCUR(12,30)
```

```
1010  WRITE(*,'(A\)')' CHANNEL='
        CALL GCUR(IROW,ICOL)
      CALL GETI(I,0,15,IROW,ICOL,2 ,'(I2\)',IRET)
        K=1
        CALL SETCUR(15,30)
        WRITE(*,'(A\)')' ON? '
        CALL GCUR(IROW,ICOL)
        K=1
      CALL YESNO(K,IROW,ICOL,IRET)
      CALL CALREL(I,K)
        GOTO 100

C     SET OFFSET ON ANALOG CARD

1500    I=0
        CALL SETCUR(12,30)
1510  WRITE(*,'(A\)')' CHANNEL='
        CALL GCUR(IROW,ICOL)
C
      CALL GETI(I,0,15,IROW,ICOL,1,'(I1\)',IRET)
        K=0
        CALL SETCUR(15,30)
      WRITE(*,'(A\)')' VALUE( -128<V<127 )'
        CALL GCUR(IROW,ICOL)
      CALL GETI(K,-128,127,IROW,ICOL,4,'(I4\)',IRET)
      CALL ZOFF(I,K)
        GOTO 100

C     READ A/D

2000    I=0
        CALL SETCUR(9,30)
2010  WRITE(*,'(A\)')' CHANNEL='
        CALL GCUR(IROW,ICOL)
      CALL GETI(I,0,15,IROW,ICOL,1,'(I1\)',IRET)
        K=0
        CALL SETCUR(12,30)
      WRITE(*,'(A\)')' GAIN='
        CALL GCUR(IROW,ICOL)
      CALL GETI(K,0,3,IROW,ICOL,1,'(I1\)',IRET)
        F=100.0
        CALL SETCUR(15,30)
      WRITE(*,'(A\)')' FREQUENCY='
        CALL GCUR(IROW,ICOL)
      CALL GETR(F,14.0,1000.,IROW,ICOL,7,'(F7.2\)',IRET)
        N=100
        CALL SETCUR(18,30)
      WRITE(*,'(A\)')' NUMBER OF POINTS='
        CALL GCUR(IROW,ICOL)
      CALL GETI(N,3,16384,IROW,ICOL,5,'(I5\)',IRET)
      AV=0
      VNSE=0
      CALL A2DONE(I,K,F,N,AV,VNSE)
        CALL SETCUR(21,30)
      WRITE(*,2400) AV

C     WRITE OUT RESULTS
```

```
        CALL SETCUR(23,30)
      WRITE(*,2450) VNSE
        GOTO 100
2400    FORMAT('AVERAGE=',F8.3\)
2450    FORMAT('RMS NOISE=',F8.5\)

C     WAIT FOR A SPECIFIED TIME

2500    F=1.0
        CALL SETCUR(12,30)
2510  WRITE(*,'(A\)')' TIME TO WAIT='
        CALL GCUR(IROW,ICOL)
      CALL GETR(F,.06,3600.,IROW,ICOL,7,'(F7.2\)',IRET)
        CALL TWAIT(F)
        GOTO 100

C     CLEAR DATA TRANSLATION BOARD

3000  CALL DTCLEAR
        GOTO 100

C     SET A/D CLOCK

3500    F=25000.0
        CALL SETCUR(12,30)
C
3510  WRITE(*,'(A\)')' DT CLOCK FREQUENCY='
        CALL GCUR(IROW,ICOL)

      CALL GETR(F,15.0,25000.,IROW,ICOL,8,'(F8.2\)',IRET)
      CALL DTCLOCK(F)
        GOTO 100

C     SET FILTER CLOCK

4000    F=150.0
        CALL SETCUR(12,30)
4010  WRITE(*,'(A\)')' CUTOFF FREQUENCY='
        CALL GCUR(IROW,ICOL)
      CALL GETR(F,20.0,150.0,IROW,ICOL,6,'(F6.2\)',IRET)
      CALL FILCLK(F)
        GOTO 100

C     RESTORE ANALOG CARDS

4500  CALL RESTOR
        GOTO 100

C     TURN REFERENCE ON/OFF

5000  CALL SETCUR(12,30)
      WRITE(*,'(A\)')'REFERENCE ON ?'
      CALL GCUR(IROW,ICOL)
      I=0
      CALL YESNO(I,IROW,ICOL,IRET)
      I=I+8
      CALL IOUTB(I,CNTRL)
      GOTO 100
```

```
C       EXIT

5500    RETURN
        END
```

IOSUBS.FOR

```
$TITLE:'I/O SUBROUTINES'
$STORAGE:2
$NOFLOATCALLS
C
C       SET CAL D/A
C
        SUBROUTINE CALDA(VOLTS)

        INTEGER*2 HIGH,V
$INCLUDE:'IOPARMS'
        K=0
        L=#00FF
        M=8
        CALL IOUTB(DAEN,CNTRL)
        CALL TWAIT(.2)
        CALL DTCLEAR
        V=NINT((VOLTS+5.0)*4096.0/10.0)
        CALL PWAIT(DTSTAT,CWAIT,K)
        CALL IOUTB(CDA,DTCOM)
        CALL PWAIT(DTSTAT,WWAIT,WWAIT)
        CALL IOUTB(K,DTDATA)
        HIGH=ISHFTR(V,M)
        LOW=IAND(L,V)
        CALL PWAIT(DTSTAT,WWAIT,WWAIT)
        CALL IOUTB(LOW,DTDATA)
        CALL PWAIT(DTSTAT,WWAIT,WWAIT)
        CALL IOUTB(HIGH,DTDATA)
        CALL PWAIT(DTSTAT,CWAIT,K)
        RETURN
        END
$PAGE
```

```
C
C      CLEAR DT BOARD
C
       SUBROUTINE DTCLEAR

$INCLUDE:'IOPARMS'
       K=0
       CALL IOUTB(CSTOP,DTCOM)
       I=INPB(DTDATA)
       CALL PWAIT(DTSTAT,CWAIT,K)
       CALL IOUTB(CCLEAR,DTCOM)
       RETURN
       END
$PAGE
```

```
C
C     SET DT CLOCK
C
      SUBROUTINE DTCLOCK(F)

$INCLUDE:'IOPARMS'
      INTEGER*4 V
      INTEGER*2 HIGH
      CALL DTCLEAR
      T=1/F*1.0E6
      M=8
      L=#00FF
      V=NINT(T/1.25-32768)+32768
      CALL PWAIT(DTSTAT,CWAIT,K)
      CALL IOUTB(CCLOCK,DTCOM)
      HIGH=ISHFTR(V,M)
      LOW=IAND(V,L)
      CALL PWAIT(DTSTAT,WWAIT,WWAIT)
      CALL IOUTB(LOW,DTDATA)
      CALL PWAIT(DTSTAT,WWAIT,WWAIT)
      CALL IOUTB(HIGH,DTDATA)
      CALL PWAIT(DTSTAT,CWAIT,K)
      RETURN
      END
$PAGE
```

```
C
C     SET CAL RELAY ON OR OFF
C     ION=0=OFF ION=1=ON
C
      SUBROUTINE CALREL(I,ION)

$INCLUDE:'IOPARMS'
      CALL IOUTB(I,AADDR)
      K=CRON
      IF( ION .EQ. 0)K=CROFF
      CALL IOUTB(K,CNTRL)
      RETURN
      END
```

```
C
C     SET OFFSET ON CHANNEL I TO IVAL
C
      SUBROUTINE ZOFF(I,IVAL)

$INCLUDE:'IOPARMS'
      CALL IOUTB(I,AADDR)
      CALL IOUTB(IVAL,ADATA)
      CALL IOUTB(ZAEN,CNTRL)
      T=.1
      CALL TWAIT(T)
      CALL IOUTB(DASON,CNTRL)
      CALL IOUTB(DASOFF,CNTRL)
      CALL IOUTB(ZDIS,CNTRL)
      RETURN
      END
$PAGE
```

```
C
C      SET UP DT A/D
C
       SUBROUTINE SETAD(AD)

       INTEGER*2 AD(5)
$INCLUDE:'IOPARMS'
       K=0
       CALL DTCLEAR
       CALL PWAIT(DTSTAT,CWAIT,K)
       CALL IOUTB(CSAD,DTCOM)
       DO 10 I=1,5
       CALL PWAIT(DTSTAT,WWAIT,WWAIT)
       CALL IOUTB(AD(I),DTDATA)
10     CONTINUE
       RETURN
       END
$PAGE
```

```
C
C      SET UP DMA CONTROLLER FOR A/D
C
       SUBROUTINE SETDMA(DM)

       INTEGER*2 DM(5)
       CALL IOUTB(#0045,11)
       CALL IOUTB(0,12)
       CALL IOUTB(DM(1),2)
       CALL IOUTB(DM(2),2)
       CALL IOUTB(DM(3),3)
       CALL IOUTB(DM(4),3)
       CALL IOUTB(DM(5),#0083)
       I=1
       CALL IOUTB(I,10)
       RETURN
       END
$PAGE
```

```
C
C     COLLECT ONE CHANNEL OF A/D
C
      SUBROUTINE A2DONE(ICHAN,IGAIN,FREQ,N,AV,VNSE)

$INCLUDE:'BUFCOM'
$INCLUDE:'IOPARMS'
      INTEGER*2 I(2),J(2),L(2),K(2),AD(5),DM(5)
      INTEGER*4 II,KK,JJ,LL,INDEX
      EQUIVALENCE (II,I),(JJ,J),(LL,L),(KK,K)
      L1=#00FF
      M=8
      CALL PHYSAD(IBUF(1),JJ)
      II=N*2
      LL=JJ+II
      IF (J(2) .EQ. L(2)) THEN
         KK=JJ
         INDEX=1
      ELSE
         K(2)=L(2)
         K(1)=0
         INDEX=(KK-JJ)/2+1
      ENDIF
      II=II-1
      DM(1)=IAND(K(1),L1)
      DM(2)=ISHFTR(K(1),M)
      DM(3)=IAND(I(1),L1)
      DM(4)=ISHFTR(I(1),M)
      DM(5)=K(2)
      AD(1)=IGAIN
      AD(2)=ICHAN
      AD(3)=ICHAN
      AD(4)=10
      AD(5)=0
      CALL DTCLOCK(FREQ)
      CALL SETDMA(DM)
      CALL SETAD(AD)
      L1=IOR(CRAD,CDMA)
      CALL PWAIT(DTSTAT,CWAIT,0)
      CALL IOUTB(L1,DTCOM)
      CALL PWAIT(DTSTAT,CWAIT,0)
      M=INPB(DTSTAT)
      M=IAND(M,#0080)
      IF (M .NE. 0)STOP 'A/D ERROR'

C     CALCULATE AVERAGE MAX AND MIN
      II=0
      MIN=IBUF(INDEX)
      MAX=MIN
      DO 100 M=0,N-1
      II=II+IBUF(INDEX+M)
      IF (IBUF(INDEX+M) .LT. MIN)MIN=IBUF(INDEX+M)
      IF (IBUF(INDEX+M) .GT. MAX)MAX=IBUF(INDEX+M)
100   CONTINUE
      R=10.0/2**IGAIN
      RR=R*2.0/4096
      AV=(II/N)
```

```
      AV=AV*RR
      AV=AV-R

C     CALCULATE RMS NOISE
      DT=1.0/FREQ
      T=N*DT
      VSQ=0.0
      DO 200 M=0,N-1
      VSQ=VSQ+(IBUF(INDEX+M)*RR-R-AV)**2*DT
200   CONTINUE
      VNSE=SQRT(1/T*VSQ)
      RETURN
      END
$PAGE
```

```
C
C      RESTORE ANALOG
C
       SUBROUTINE RESTOR

$INCLUDE:'SETCOM'
$INCLUDE:'IOPARMS'
       CALL IOUTB(DASOFF,CNTRL)
       CALL IOUTB(SHOFF,CNTRL)
       DO 100 I=0,15
       J=I+1
100    CALL ZOFF(I,IOFFS(J))
       RETURN
       END
$PAGE
```

```
C
C      SET FILTER CLOCK
C
       SUBROUTINE FILCLK(F)

       INTEGER*2 C(2),HIGH
       INTEGER*4 COUNT
       EQUIVALENCE (COUNT,C)
$INCLUDE:'IOPARMS'
       COUNT=1.193182E6/100.0/F
C      DISARM COUNTER
       CALL IOUTB(#D0,TIMERC)
C      LOAD COUNTER
       CALL IOUTB(#0D,TIMERC)
       LOW=IAND(#00FF,C(1))
       HIGH=ISHFTR(C(1),8)
       CALL IOUTB(LOW,TIMERD)
       CALL IOUTB(HIGH,TIMERD)
C      START COUNTER 5
       CALL IOUTB(#70,TIMERC)
       RETURN
       END
$PAGE
```

```
C
C     INITIALIZE I/O
C
      SUBROUTINE INITIO

$INCLUDE:'IOPARMS'
C     SET UP 8255 CHIP
      CALL IOUTB(#90,CNTRL)
C     SET UP ANALOG CONTROL LINES
      CALL IOUTB(0,IPC)
      CALL IOUTB(DASOFF,CNTRL)
      CALL RESTOR

C     SET UP TIMER CHIP MASTER MODE
      CALL IOUTB(#0FF,TIMERC)
      CALL IOUTB(#5F,TIMERC)
      CALL IOUTB(#DF,TIMERC)
      CALL IOUTB(#E8,TIMERC)
      CALL IOUTB(#17,TIMERC)
      CALL IOUTB(#D0,TIMERD)
      CALL IOUTB(#49,TIMERD)

C     SET UP COUNTER 5
      CALL IOUTB(1,TIMERC)
      CALL IOUTB(#21,TIMERD)
      CALL IOUTB(2,TIMERD)
      CALL IOUTB(9,TIMERC)
      CALL IOUTB(4,TIMERD)
      CALL IOUTB(0,TIMERD)

C     SET UP COUNTER 5(FILTER CLOCK)
      CALL IOUTB(5,TIMERC)
      CALL IOUTB(#22,TIMERD)
      CALL IOUTB(#0B,TIMERD)
      CALL IOUTB(#0D,TIMERC)
      CALL IOUTB(119,TIMERD)
      CALL IOUTB(0,TIMERD)

C     START COUNTER 5
      CALL IOUTB(#70,TIMERC)
      RETURN
      END
```

LOADTAPE.FOR

```
$STORAGE:2
$NOFLOATCALLS

      SUBROUTINE LOADT

$INCLUDE:'BUFCOM'
$INCLUDE:'STATCOM'
$INCLUDE:'SETCOM'
      INTEGER*2 ISTAT
      LOGICAL IEXIST
      CHARACTER*16 FN
      CALL CLRSCR

      CALL SETCUR(0,0)

C     IS A TAPE ALREADY LOADED?
      IF(TINIT .EQ. 1)THEN
      WRITE(*,'(A\)')'A TAPE IS ALREADY LOADED'
      GOTO 1000
      ENDIF

C     MAKE SURE TAPE IS READY
      WRITE(*,'(A\)')'IS THE TAPE READY?'
      I=0
10    CALL YESNO(I,0,18,IRET)
      IF( I .EQ. 0) GOTO 10
      CALL SETCUR(1,0)

C     CHECK DRIVE D FOR NAME.VOL
      FN='D:NAME.VOL '
      INQUIRE(FILE=FN,EXIST=IEXIST)
C     IF (I .NE. 0) THEN
C     WRITE(*,'(A\)')'ERROR READING DRIVE D'
C     GOTO 1000
C     ENDIF
      CALL SETCUR(1,0)
      IF(IEXIST)THEN

C     READ IN VOLUME INFORMATION
       OPEN(9,FILE=FN)
       READ(9,8000)TVOL,(IBUF(I),I=1,100)
8000   FORMAT(A56,100I7)
       CLOSE(9)
       TFILE=LFILE
       WRITE(*,9000)TNAME,TCDATE,TCTIME
9000   FORMAT('TAPE NAME IS ',A8,' CREATED ',A8,1X,A8\)
       CALL SETCUR(3,0)
       WRITE(*,9010)LFILE,TLDATE,TLTIME
9010   FORMAT('LAST FILE IS ',A16,' CREATED ',A8,1X,A8\)
      ELSE

C       NEW TAPE GET NAME AND CREATE NAME.VOL

       TNAME='        '
```

```
      WRITE(*,'(A\)')'NEW TAPE--INPUT NAME '
      CALL GETSTR(TNAME,8,1,21,IRET)
      OPEN(9,FILE=FN,STATUS='NEW')

C     GET DATE AND TIME
      CALL GDATE(I,J,K)
      I=I-1900
      WRITE(TCDATE,9020)J,K,I
9020  FORMAT(I2,'-',I2,'-',I2)
      TLDATE=TCDATE
      CALL GTIME(I,J,K)
      WRITE(TCTIME,9030)I,J,K
9030  FORMAT(I2,':',I2,':',I2)
      TLTIME=TCTIME
      LFILE='D:NEWFILE.DTA     '
      TFILE=LFILE
      WRITE(9,8000)TVOL,(IBUF(I),I=1,100)
      CLOSE(9)
     ENDIF
     TINIT=1

C    FLUSH BUFFERS AND EXIT
     CALL TAPE(3,4,ISTAT)
1000 CALL WAITKY
     RETURN
     END
```

LOGO.FOR

```
$STORAGE:2
$NOFLOATCALLS
C
C     SUBROUTINE LOGO- DRAWS OPENING LOGO USING FILE "LOGO"
C
      SUBROUTINE LOGO

      CHARACTER*80 STR1
      CHARACTER*13 F
      F='/HALO107.FNT/'
      CALL INITGR
      CALL SETIEEE(1)
      CALL SETFONT(F)
      CALL SETWORLD(0.,0.,1000.,1000.)

      CALL SETLNW(3)
      CALL BOX(5.,5.,995.,995.)
      CALL PTABS (30.,360.)
      CALL LNABS (970.,360.)
      CALL PTABS (30.,122.)
      CALL LNABS (970.,122.)

      OPEN (9,FILE='LOGO')
      READ (9,'(I2)') NSTR
      DO 10 I = 1,NSTR
        READ (9,'(I4,3F9.2,A)')LW,HT,X,Y,STR1
        CALL SETLNW(LW)
        CALL MOVTCA (X,Y)
        CALL SETSTEXT (HT,1.,0)
        CALL STEXT(STR1)
10    CONTINUE

      CALL DELTCUR
      CALL KCLEAR
20    J=IGKEY()
      IF (J .EQ. 80 .OR. J .EQ. 112) THEN
        CALL SETGPR(1)
        CALL GPRINT
        GO TO 20
      END IF
      IF( J .EQ. 0)GOTO 20
      CALL CLOSEGR
      RETURN
      END
```

LOPASS.FOR

```
$TITLE:'SUBROUTINE LOPASS'
$NOFLOATCALLS
$STORAGE:2
**********************************************************************
        SUBROUTINE LOPASS (ARRAY, NCH, NS, MOVAV1, MOVAV2)
**********************************************************************
*  This subroutine filters a signal with a lo-pass filter.
*
*  <-> ARRAY  real*4     2-D Input array.  Channel 1 is filtered.
*                        The data should start at position 2 and
*                        continue to NS + 1.
*                        The output starts in position 1, and
*                        corresponds to what used to be the MOVAV1-th
*                        point.  (The array gets shifted.)
*  --> NCH    integer*4  1st dimension of ARRAY. (# of channels.)
*  --> NS     integer*4  no. of samples in ARRAY.
*  --> MOVAV1 integer*4  no. of points in moving average,
*  --> MOVAV2 integer*4  no. of points to center of moving average
*                        (MOVAV1 / 2),
**********************************************************************
$LARGE: ARRAY
      INTEGER*4 MOVAV1, MOVAV2, NCH, NS, I, I1, I2, M1, M2, N
      REAL*4 ARRAY (*), SCM1
*
*  Initialize moving average.
*
      ARRAY (1) = 0
      I1 = 1
      DO 40 I=1, MOVAV1
        I1 = I1 + NCH
        ARRAY (1) = ARRAY (1) + ARRAY (I1)
   40 CONTINUE
      ARRAY (1) = ARRAY (1) / MOVAV1
*
*  Filter signal.
*
      I1 = 1
      I2 = I1 + NCH
      SCM1 = 1. / MOVAV1
      M1 = MOVAV1 * NCH
      M2 = MOVAV2 * NCH
*
      DO 50 I = 2, NS
        ARRAY (I2) = ARRAY (I1) + SCM1 * (ARRAY (I2 + M1) -
     &                ARRAY (I2))
        I1 = I2
        I2 = I2 + NCH
   50 CONTINUE
      RETURN
      END
```

```
C
C     ROAD MEASUREMENT SUBROUTINE
C
```

MEASURE.FOR

```
$TITLE:'MEASURE'
$STORAGE:2
$NOFLOATCALLS

      SUBROUTINE MEASURE

$INCLUDE:'STATCOM'
      CHARACTER*32 IMENU(8)
      INTEGER*2 MA(8)
      MI=8
      DO 10 I=1,MI
10    MA(I)=1

      IMENU(1)='MAKE ROAD MEASUREMENTS'
      IMENU(2)='SELECT CONFIGURATION'
      IMENU(3)='DO ELECTRICAL CALIBRATION'
      IMENU(4)='DO BOUNCE TEST'
      IMENU(5)='CHECK PULSER'
      IMENU(6)='MEASURE ROAD'
      IMENU(7)='PROCESS DATA'
      IMENU(8)='EXIT TO MAIN MENU'

C     SET DEFAULT TO MEASURE ROAD

C     GET SELECTION
50    IF( CALYN .EQ. 0)THEN
       IDEF=2
      ELSEIF (BOUNYN .EQ. 0)THEN
       IDEF=3
      ELSEIF (PULYN .EQ.0)THEN
       IDEF=4
      ELSE
       IDEF=5
      ENDIF
      CALL MENU(IMENU,MI,MA,IDEF,IRET)
      CALL CLRSCR
      GOTO (100,200,300,400,500,600,700)IRET

C     CONFIGURE SYSTEM

100   CALL CONFIGURE
      GOTO 50

C     DO ELECTRICAL CAL
200   CALL CALIB
      GOTO 50

C     DO BOUNCE TEST
300   CALL TEST(1)
      BOUNYN=1
      GOTO 50
```

```
C       CHECK PULSER
400     CALL PULSE
        PULYN=1
        GOTO 50

C       MEASURE ROAD

500     CALL TEST(0)
        GOTO 50

C       PROCESS DATA
600     CALL PROCESS
        GOTO 50

C       RETURN TO MAIN PROGRAM
700     RETURN
        END
```

C MINV.FOR

```
NAASA  2.1.020 MINV     FTN   06-24-75     THE UNIV OF MICH COMP CTR
C     Modified July 9, 1986 so that it will compile as Fortran 77.
C     ...............................................................
C
C        SUBROUTINE MINV
C
C        PURPOSE
C           INVERT A MATRIX
C
C        USAGE
C           CALL MINV(A,N,D,L,M)
C
C        DESCRIPTION OF PARAMETERS
C           A - INPUT MATRIX, DESTROYED IN COMPUTATION AND REPLACED BY
C               RESULTANT INVERSE.
C           N - ORDER OF MATRIX A
C           D - RESULTANT DETERMINANT
C           L - WORK VECTOR OF LENGTH N
C           M - WORK VECTOR OF LENGTH N
C
C        REMARKS
C           MATRIX A MUST BE A GENERAL MATRIX
C
C        SUBROUTINES AND FUNCTION SUBPROGRAMS REQUIRED
C           NONE
C
C        METHOD
C           THE STANDARD GAUSS-JORDAN METHOD IS USED. THE DETERMINANT
C           IS ALSO CALCULATED. A DETERMINANT OF ZERO INDICATES THAT
C           THE MATRIX IS SINGULAR.
C
C     ...............................................................
C
$STORAGE:2
$NOFLOATCALLS
      SUBROUTINE MINV(A, N, D, L, M)

      DIMENSION A(*), L(*), M(*)
C
C        ............................................................
C
C        IF A DOUBLE PRECISION VERSION OF THIS ROUTINE IS DESIRED, THE
C        C IN COLUMN 1 SHOULD BE REMOVED FROM THE DOUBLE PRECISION
C        STATEMENT WHICH FOLLOWS.
C
C     DOUBLE PRECISION A,D,BIGA,HOLD
C
C        THE C MUST ALSO BE REMOVED FROM DOUBLE PRECISION STATEMENTS
C        APPEARING IN OTHER ROUTINES USED IN CONJUNCTION WITH THIS
C        ROUTINE.
C
C        THE DOUBLE PRECISION VERSION OF THIS SUBROUTINE MUST ALSO
C        CONTAIN DOUBLE PRECISION FORTRAN FUNCTIONS.  ABS IN STATEMENT
C        10 MUST BE CHANGED TO DABS.
C
C        ............................................................
```

```
C
C         SEARCH FOR LARGEST ELEMENT
C
      D = 1.0
      NK = -N
      DO 190 K = 1, N
        NK = NK + N
        L(K) = K
        M(K) = K
        KK = NK + K
        BIGA = A(KK)
        DO 30 J = K, N
          IZ = N * (J - 1)
          DO 30 I = K, N
            IJ = IZ + I
   10       IF (ABS(BIGA) - ABS(A(IJ))) 20, 30, 30
   20       BIGA = A(IJ)
            L(K) = I
            M(K) = J
   30   CONTINUE
C
C         INTERCHANGE ROWS
C
        J = L(K)
        IF (J - K) 60, 60, 40
   40   KI = K - N
        DO 50 I = 1, N
          KI = KI + N
          HOLD = -A(KI)
          JI = KI - K + J
          A(KI) = A(JI)
   50   A(JI) = HOLD
C
C         INTERCHANGE COLUMNS
C
   60   I = M(K)
        IF (I - K) 90, 90, 70
   70   JP = N * (I - 1)
        DO 80 J = 1, N
          JK = NK + J
          JI = JP + J
          HOLD = -A(JK)
          A(JK) = A(JI)
   80   A(JI) = HOLD
C
C         DIVIDE COLUMN BY MINUS PIVOT (VALUE OF PIVOT ELEMENT IS
C         CONTAINED IN BIGA)
C
   90   IF (BIGA) 110, 100, 110
  100   D = 0.0
        RETURN
  110   DO 130 I = 1, N
          IF (I - K) 120, 130, 120
  120     IK = NK + I
          A(IK) = A(IK) / (-BIGA)
  130   CONTINUE
C
C         REDUCE MATRIX
```

```
C
      DO 160 I = 1, N
        IK = NK + I
        HOLD = A(IK)
        IJ = I - N
        DO 160 J = 1, N
          IJ = IJ + N
          IF (I - K) 140, 160, 140
140       IF (J - K) 150, 160, 150
150       KJ = IJ - I + K
          A(IJ) = HOLD * A(KJ) + A(IJ)
160   CONTINUE
C
C      DIVIDE ROW BY PIVOT
C
      KJ = K - N
      DO 180 J = 1, N
        KJ = KJ + N
        IF (J - K) 170, 180, 170
170     A(KJ) = A(KJ) / BIGA
180   CONTINUE
C
C      PRODUCT OF PIVOTS
C
      D = D * BIGA
C
C      REPLACE PIVOT BY RECIPROCAL
C
      A(KK) = 1.0 / BIGA
190 CONTINUE
C
C      FINAL ROW AND COLUMN INTERCHANGE
C
    K = N
200 K = (K - 1)
    IF (K) 270, 270, 210
210 I = L(K)
    IF (I - K) 240, 240, 220
220 JQ = N * (K - 1)
    JR = N * (I - 1)
    DO 230 J = 1, N
      JK = JQ + J
      HOLD = A(JK)
      JI = JR + J
      A(JK) = -A(JI)
230 A(JI) = HOLD
240 J = M(K)
    IF (J - K) 200, 200, 250
250 KI = K - N
    DO 260 I = 1, N
      KI = KI + N
      HOLD = A(KI)
      JI = KI - K + J
      A(KI) = -A(JI)
260 A(JI) = HOLD
    GO TO 200
270 RETURN
    END
```

PLOT.FOR

```
$TITLE:'THE PLOT SUBROUTINE'
$STORAGE:2
$NOFLOATCALLS

      SUBROUTINE PLOT(MODE,IACTIV,NCHAN,NPTS,ICH,IIS,ITOT,
     &              DX,XMIN,XMAX,XSTART,KCURS,YMIN,YMAX,NAME,UNITS,
     &              XNAME,XUNITS,GAIN,OFF,IUPDT, ISTART, NPTOT,
     &              NPMAX, TITLE)
**********************************************************************
* --> MODE   int*2   DATA TYPE 0=INTEGER 1=FLOATING POINT
* <-> IACTIV int*2   the active plot (1 or 2)
* --> NCHAN  int*2   number of channels to be plotted (1 or 2).
* --> NPTS   int*4   number of points (per channel) to plot
* --> ICH    int*2   array with id nos. of the channel(s) being plotted.
* --> IIS    int*4   offset (in array) to first point to be plotted.
* --> ITOT   int*2   number of channels in buffer.
* --> DX     real*4  sample interval. (x axis gain)
* <-> XMIN   real*4  minimum limit for x values.
* <-> XMAX   real*4  maximum limit for x values.
* --> XSTART real*4  value of x at start of file (i=0).
* <-> KCURS  int*4   offset in file to cursor position (0=1st sample).
* <-> YMIN   real*4  array of min y limits for all (ITOT) channels.
* <-> YMAX   real*4  array of max y limits for all (ITOT) channels.
* --> NAME   char*8  array of names of all channels.
* --> UNITS  char*8  array of names of units of all of the channels.
* --> XNAME  char*8  name of variable plotted on the x axis.
* --> XUNITS char*8  name of units for variable plotted on the x axis.
* --> GAIN   real*4  array of channel gains used for integer data.
* --> OFF    real*4  array of offsets of channels used for integer data.
* <-> IUPDT  int*2   0=don't redraw; 1=rescale y axis on active plot;
*                    2=both plots, 3=quit.  The only values on exit
*                    are 2 and 3.
* <-> ISTART int*4   offset (in file) to first point in plot array.
* --> NPTOT  int*4   number of samples in file.
* --> NPMAX  int*4   max number of points that can be plotted.
* --> FNAME  char*30 title for plots.
**********************************************************************
$INCLUDE:'BUFCOM'

      INTEGER*2 ICH(2)
      DIMENSION YMIN(*),YMAX(*),XPUL(2),XPLR(2),YPUL(2),YPLR(2)
      DIMENSION RDATA(65536),GAIN(2),OFF(2),YTUL(2),YTLR(2)
      DIMENSION XYUL(2),YYUL(2),XYLR(2),YYLR(2),XTUL(2),XTLR(2)
      DIMENSION XXUL(2),YXUL(2),XXLR(2),YXLR(2)
      EQUIVALENCE (IBUF,RDATA)
      INTEGER*4 J,KCURS,NPTS,ISTART,JJ,KK, IIS, NPTOT,NPMAX
      CHARACTER*10 S2(2),S4,STRING
      CHARACTER*11 S1(2),S3
      CHARACTER*8 UNITS(*),NAME(*),XUNITS,XNAME
      CHARACTER*12 S(2),SX(2)
      CHARACTER*30 TITLE
      CHARACTER*32 ST

C     SET UP VIEWPORT COORDS FOR PLOTS
```

```
      DO 10 I=1,2
        XPUL(I)=.1
        YPUL(I)=.005
        XPLR(I)=.995
        YPLR(I)=.88

C     SET UP VIEWPORT COORDS FOR Y-AXES
        XYUL(I)=0.
        YYUL(I)=.005
        XYLR(I)=.098
        YYLR(I)=.88

C     SET UP VIEWPORT COORDS FOR X-AXES
        XXUL(I)=.1
        YXUL(I)=.90
        XXLR(I)=.995
        YXLR(I)=.93

C     SET UP VIEWPORT COORDS FOR TEXT
        XTUL(I)=.005
        YTUL(I)=.95
        XTLR(I)=.995
        YTLR(I)=.985
10    CONTINUE

      IF(NCHAN .EQ. 2) THEN
        YPUL(2)=.545
        YYUL(2)=.545
        YPLR(1)=.38
        YYLR(1)=.38
        YPUL(1)=.045
        YYUL(1)=.045
        YXUL(1)=.40
        YXLR(1)=.43
        YTUL(1)=.45
        YTLR(1)=.495
      ENDIF

*  Set the title and x-axis Halo text strings

      WRITE (ST,'(''\'',A30,''\'')') TITLE
      WRITE(S3,9010)XNAME
      WRITE(S4,9020)XUNITS
9010  FORMAT('\',A8,':\')
9020  FORMAT('\',A8,'\')

*  Loop to do 1 or 2 plots on screen.

      IF (IUPDT .EQ. 0) GO TO 110
      CALL SETLNW(1)
   20 DO 100 I=1,NCHAN

*  Skip a plot if it's not active and if IUPDT=1

        IF (IUPDT .EQ. 1 .AND. I .NE. IACTIV) GO TO 100
        ICHI = ICH (I)

*  Determine tick spacing on x axis.  Label it and put in fasttext
```

```
*  unless IUPDT = 1

        TX=(XMAX-XMIN)/10.
        CALL SCLUP(TX,T,XTICK)
        CALL TIKSET (XMIN, XMAX, XTICK, XTMIN, XTMAX, NXTICK)

        IF (IUPDT .NE. 1) THEN

          WRITE(S1(I),9010)NAME(ICHI)
          WRITE(S2(I),9020)UNITS(ICHI)

          CALL FTLOCATE (34*I/NCHAN, 2)
          CALL FTEXT (ST)
          CALL FTLOCATE (34*I/NCHAN,33)
          CALL FTEXT(S1(I))
          CALL FTLOCATE (34*I/NCHAN,53)
          CALL FTEXT(S2(I))
          CALL FTLOCATE (34*I/NCHAN,63)
          CALL FTEXT(S3)
          CALL FTLOCATE (34*I/NCHAN,83)
          CALL FTEXT(S4)

          SCLXLB = (XMAX - XMIN) / 2 / (90 * (XXLR (I) - XXUL (I)))
          X = XXUL(I)   - .07
          CALL SETVIEW(X,YXUL(I),XXLR(I),YXLR(I),-1,0)
          X = XMIN - .07 * (XMAX - XMIN) /  (XXLR (I) - XXUL (I))
          CALL SETWORLD(X,0.,XMAX,1.)
          DO 30 K=0, NXTICK - 1
            T=XTICK*K+XTMIN
            IF (ABS (T) .LT. .01 * XTICK) T = 0
            L = 8
              CALL LABEL (T, STRING, L)
              T = T - L * SCLXLB
            XMAX2 = XMAX - (2 * L + 1) * SCLXLB
              IF (T .GT. XMAX2) T = XMAX2
            CALL MOVTCA(T, 0)
            CALL TEXT(STRING)
30        CONTINUE
        END IF

*  Determine tick spacing for the y axis and label it.

        TY=(YMAX(ICHI)-YMIN(ICHI))/10.*NCHAN
        CALL SCLUP(TY,T,YTICK)
        CALL TIKSET (YMIN(ICHI), YMAX(ICHI), YTICK, YTMIN,
     &                    YTMAX, NYTICK)

        CALL SETVIEW(XYUL(I),YYUL(I),XYLR(I),YYLR(I),-1,0)
        CALL SETWORLD(0.,YMIN(ICHI),10.,YMAX(ICHI))

        YLBOFF = (YMAX(ICHI) - YMIN(ICHI)) * NCHAN / 60
        DO 40 K = 0, NYTICK - 1
          T=YTICK * K + YTMIN
          IF (ABS (T) .LT. .01 * YTICK) T = 0
          L = 8
          CALL LABEL(T,STRING,L)
          X = 9 - L
          T = T - YLBOFF
```

```
          CALL MOVTCA (X, T)
          CALL TEXT(STRING)
40      CONTINUE

*   Open the viewport for the data, and draw the grids.

        CALL SETVIEW (XTUL(I), YYUL(I), XTLR(I), YTLR(I), 1, -1)
        CALL SETVIEW(XPUL(I),YPUL(I),XPLR(I),YPLR(I),1,0)
        CALL SETWORLD(XMIN,YMIN(ICHI),XMAX,YMAX(ICHI))
        CALL SETLNST(2)

        DO 50 K = 0, NXTICK - 1
          T=XTICK*K+XTMIN
          IF (T .NE. XMIN .AND. T .NE. XMAX) THEN
            CALL MOVABS(T,YMIN(ICHI))
            CALL LNABS(T,YMAX(ICHI))
          END IF
50      CONTINUE

        DO 60 K=0, NYTICK - 1
          T=YTICK * K + YTMIN
          IF (T .NE. YMIN(ICHI) .AND. T .NE. YMAX(ICHI)) THEN
            CALL MOVABS(XMIN,T)
            CALL LNABS(XMAX,T)
          END IF
60      CONTINUE

*   Now plot the data.

        CALL SETLNST(1)
        KK=IIS
        IF (MODE .EQ. 0)THEN
          Y=FLOAT(IBUF(KK+ICHI))*GAIN(ICHI)-OFF(ICHI)
        ELSE
          Y=RDATA(KK+ICHI)
        ENDIF
        CALL PTABS(XMIN,Y)

*   Plot'm up.

        DO 70 J=0,NPTS-1
          X=DX*J+XMIN
          IF (MODE .EQ. 0)THEN
            Y=FLOAT(IBUF(KK+ICHI))*GAIN(ICHI)-OFF(ICHI)
          ELSE
            Y=RDATA(KK+ICHI)
          ENDIF
          KK=KK+ITOT
70      CALL LNABS(X,Y)

100   CONTINUE
110     CONTINUE

*   Update cursor coordinates.

200   JJ=(KCURS-ISTART)*ITOT+IIS
      X = (KCURS-ISTART) * DX + XMIN
      I = IACTIV
```

```
      ICHI = ICH(I)
      IF (MODE .EQ. 0)THEN
        Y=FLOAT(IBUF(JJ+ICHI))*GAIN(ICHI)-OFF(ICHI)
      ELSE
        Y=RDATA(JJ+ICHI)
      ENDIF

*   Write cursor position.

      WRITE(S(I),9000)Y
9000  FORMAT('\',F10.4,'\')
      CALL FTLOCATE (34*I/NCHAN, 42)
      CALL FTEXT(S(I))
      CALL FTLOCATE (34*I/NCHAN, 72)
      WRITE(SX(I),9000)X
      CALL FTEXT(SX(I))

*   Re-draw the cursor.

      CALL SETVIEW(XPUL(I),YPUL(I),XPLR(I),YPLR(I),-1,-1)
      CALL SETWORLD(XMIN,YMIN(ICHI),XMAX,YMAX(ICHI))
      CURX=50.*(XMAX-XMIN)/720.
      CURY=50.*NCHAN*(YMAX(ICHI)-YMIN(ICHI))/348.
      CALL INITHC(CURY,CURX,1)
      CALL MOVHCA(X,Y)

*   Get next cursor position.

      CALL GRCURS (ISTART, IACTIV, KCURS, NPTS, NCHAN, NPTOT,
     &       NPMAX, IUPDT, XMIN, XMAX, XSTART, DX, YMIN, YMAX, ICH)

      IF (IUPDT .EQ. 1) GO TO 20
      IF(IUPDT .EQ. 0) GOTO 200
      RETURN
      END
```

PLOTELV.FOR

```
$TITLE:'SUBROUTINE PLTELV'
$STORAGE:2
$NOFLOATCALLS
      SUBROUTINE PLTELV (HANDLE, QNDPLT)
*******************************************************************
*  --> HANDLE int*2  handle to data file.
*  --> QNDPLT log    .true. if its a quick and dirty plot.
*******************************************************************
$INCLUDE:'BUFCOM'
$INCLUDE:'SETCOM'
$INCLUDE:'STATCOM'
$INCLUDE:'HANDLES'
      INTEGER*2 IOF(2),ICHAN(2),IPTR(8), MA(15)
      CHARACTER*1 DR
      CHARACTER*3 EXT
      CHARACTER*8 N(8),U(8),XNAME,XUNITS, FN,STR1
      CHARACTER*30 TITLE
      CHARACTER*32 IMENU (15), BSTITL
      CHARACTER*60 BSPRMT
      LOGICAL QNDPLT
      REAL*4 YMIN(8),YMAX(8),YRANGE(8), YMXRNG (8),BASES (12)
      INTEGER*4 NPTS,ISTART,NPMAX,II,JJ,MBYTES,IP,IO,LBYTES,IIS,
     &            KCURS,NPTOT,MOVAV1, MOVAV2

      DATA BSTITL /'BASELENGTH TO REMOVE LONG WAVES'/
      DATA BSPRMT /'BASELENGTH:'/

      XNAME=CHID(10)
      XUNITS=UNITS(10)
      MAXBUF = MXBFSZ

*  Set sample interval and number of points

      IF (QNDPLT) THEN
        NPTOT = NSRTOT
        DX = DXTRIM
      ELSE
        NPTOT = NSPTOT
        DX = DELTAX
      END IF

*  Get baselength for moving average.

      IF (TSTTYP .EQ. 2) THEN
        CALL GETLEN (FLTBAS, DX * 5., LNGWAV * 4., XUNITS, BSTITL,
     &            BSPRMT, IRET)
        IF (IRET .EQ. 1) RETURN

        MOVAV1 = FLTBAS / DX + 1
        MOVAV2 = MOVAV1 / 2 + 1
      ELSE
        MOVAV1 = 0
        MOVAV2 = 0
      END IF
```

```
*  Set limits and constants derived from the baselength.

      NPMAX = MAXBUF / NCHPRF - MOVAV1 - TRIM
      IF (NPMAX .GT. NPTOT + 1) NPMAX = NPTOT + 1

*  Create title to pass to PLOT.

      CALL FNMAKE (DR, FN, EXT, PFILE, 1)
      TITLE = DR
      TITLE(2:2) = ':'
      TITLE(3:) = FN
      TITLE(13:) = 'FLT. BASE '
      L = 8
      CALL STRX (FLTBAS, STR1, L)
      TITLE (23:) = STR1(:L)
      IF (TSTTYP .EQ. 6) TITLE (13:) = 'BOUNCE'

C     PUT CHANNEL INFO INTO ARRAYS FOR PLTSEL CALL

      IF (LPROF) THEN
        N(ILPRF) = 'L. ELEV'
        U(ILPRF) = UNITS(1)
      END IF

      IF (RPROF) THEN
        N(IRPRF) = 'R. ELEV'
        U(IRPRF) = UNITS(1)
      END IF

      N (3) = 'PROFILES'

*  Set up default values for PLTSEL

      NCH = NCHPRF
      DO 15 I = 1, NCH
        YMXRNG (I) = FLTBAS * .1
        IF (TSTTYP .EQ. 6) YMXRNG (I) = 2048 * GAIN (1)
   15   CALL SCLUP (YMXRNG(I),X1,YMXRNG(I))

      IOF (1) = 1
      IOF (2) = 2
      IF (XRANGE .LE. 5. * DX) XRANGE = 5. * DX
      IF (XRANGE .GT. DX * NPTOT) XRANGE = DX * NPTOT
      KCURS = XCURS / DX

C     SELECT CHANNEL(S) AND SCALE

      CALL PLTSEL(NCH, N, U, XNAME, XUNITS, DX, XMIN, 0.,
     &              XRANGE, YRANGE, YMXRNG, NPTS, NPMAX, NPTOT,
     &              KCURS,IOF)

      IF(IOF (1) .EQ. 0 .AND. IOF (2) .EQ. 0) RETURN
      NCH = 1
      IF (IOF (2) .NE. 0) NCH = 2
      XCURS = KCURS * DX

C     SET UP HALO
```

```
      CALL INITGR
      CALL SETIEEE(1)
      CALL FTSIZE (1,10)
      CALL FTCOLOR (1,0)
      CALL FTINIT

      DO 19 I=1,NCH
        J = IOF(I)
        YMAX (J) = YRANGE (J)
        YMIN (J) = -YRANGE (J)
   19 CONTINUE
      XMAX = XMIN + XRANGE

      NCHTOT = NCHPRF
      IACTIV = 1
      IUPDT = 2

   20 ISTART=XMIN/DX
      IF(NPTS .GT. NPTOT-ISTART) NPTS=NPTOT-ISTART
      IIS = 0

C     READ IN DATA

100   CALL GETELV (ISTART, NPTS, MOVAV1, MOVAV2, QNDPLT, HANDLE,
     &               IERR)

C     PLOT
      CALL PLOT (1,IACTIV,NCH,NPTS,IOF,IIS,NCHTOT,DX,XMIN,XMAX,0.,
     &       KCURS,YMIN,YMAX,N,U,XNAME,XUNITS,G,O,IUPDT,ISTART,NPTOT,
     &       NPMAX,TITLE)

      IF(IUPDT .EQ. 3)THEN
        XCURS = KCURS * DX
        XRANGE = XMAX - XMIN
        CALL CLOSEGR
        RETURN
      END IF

      GOTO 20
      END
```

PLOTRAW.FOR

```
$TITLE:'RAW DATA PLOT'
$STORAGE:2
$NOFLOATCALLS


      SUBROUTINE PLTRAW (HANDLE)
******************************************************************
$INCLUDE:'BUFCOM'
$INCLUDE:'SETCOM'
$INCLUDE:'STATCOM'
$INCLUDE:'HANDLES'
      INTEGER*2 IOF(2),ICHAN(2),IPTR(8)
      CHARACTER*1 DR
      CHARACTER*3 EXT
      CHARACTER*8 N(8),U(8),XNAME,XUNITS, FN
      CHARACTER*30 TITLE
      REAL*4 YMIN(8),YMAX(8),G(8),O(8), YRANGE(8), YMXRNG (8)
      INTEGER*4 NPTS,ISTART,NPMAX,II,JJ,MBYTES,IP,IO,LBYTES,IIS,
     &          KCURS,NPTOT
      CALL CLRSCR

      NPMAX=MXBFSZ * 2 / NCHRAW
      NPTOT = PASSA
      IF (NPMAX .GT. NPTOT) NPMAX = NPTOT
      CALL FNMAKE (DR, FN, EXT, PFILE, 1)
      TITLE = 'RAW DATA FROM FILE:  :'
      TITLE(21:21) = DR
      TITLE(23:) = FN

C     GET XUNITS,XNAME,AND DELTAX
      IF(IDMODE .EQ. #0B21)THEN
C     TIME BASED SAMPLING
       XNAME='TIME'
       XUNITS='SECONDS'
       DELTAX=IDIV*.4190477E-6
      ELSE
C     DISTANCE BASED SAMPLING
       XNAME=CHID(10)
       XUNITS=UNITS(10)
      ENDIF

C     PUT CHANNEL INFO INTO ARRAYS FOR PLTSEL CALL
      L=ADSTRT
      DO 10 I=1,NCHAN
        M=L+1
        N(I)=CHID(M)
        U(I)=UNITS(M)
        YMXRNG (I) = ABS(GAIN(M)) * 2048.
        CALL SCLUP (YMXRNG (I),X1, YMXRNG(I))
        IPTR(I)=M
        L=L+1
        IF( L .GT. 7)L=0
10    CONTINUE
```

```
*  Set up default values for PLTSEL

      IOF (1) = 1
      IOF (2) = 2
      IF (XRANGE .GT. DELTAX * PASSA) XRANGE = 10.
      KCURS = XCURS / DELTAX
      IF (KCURS .LT. 0 .OR. KCURS .GT. PASSA) KCURS = 1

C     SELECT CHANNEL(S) AND SCALE

11    CALL CLRSCR
      CALL PLTSEL(NCHAN, N, U, XNAME, XUNITS, DELTAX, XMIN,0.,
     &       XRANGE,YRANGE, YMXRNG, NPTS, NPMAX, NSAMP,KCURS,IOF)

      IF(IOF (1) .EQ. 0 .AND. IOF (2) .EQ. 0) RETURN
      NCH = 1
      IF (IOF (2) .NE. 0) NCH = 2
      XCURS = KCURS * DELTAX

C     SET UP HALO
      CALL INITGR
      CALL SETIEEE(1)
      CALL FTSIZE (1,10)
      CALL FTCOLOR (1,0)
      CALL FTINIT

      DO 15 I=1,NCH
        J = IOF(I)
        G(J)=GAIN(IPTR(J))
        O(J)=ZDATA(IPTR(J))
        YMAX (J) =  YRANGE (J)
        YMIN (J) = - YRANGE (J)
        IF (J .EQ. ICHV) YMIN (J) = 0
   15 CONTINUE
      XMAX = XMIN + XRANGE

      IACTIV = 1
      IUPDT = 2
   20 ISTART=XMIN/DELTAX
      IF(NPTS .GT. NPTOT-ISTART) NPTS=NPTOT-ISTART
      IIS = 0

C     READ IN DATA

100   OFFSET=ISTART*2*NCHAN
      BYTES=NPTS*2*NCHAN
      CALL RDTAPE(HANDLE,IBUF,OFFSET,BYTES,IER)

C     PLOT
      CALL PLOT (0,IACTIV,NCH,NPTS,IOF,IIS,NCHAN,DELTAX,XMIN,XMAX,
     &      0.,KCURS,YMIN,YMAX,N,U,XNAME,XUNITS,G,O,IUPDT,ISTART,
     &      NPTOT,NPMAX,TITLE)

      IF(IUPDT .EQ. 3)THEN
        XCURS = KCURS * DELTAX
        XRANGE = XMAX - XMIN
        CALL CLOSEGR
        RETURN
```

```
END IF

GOTO 20
END
```

PLOTRUT.FOR

```
$TITLE:'SUBROUTINE PLTRUT'
$STORAGE:2
$NOFLOATCALLS

      SUBROUTINE PLTRUT (HANDLE)
*******************************************************************
*  --> HANDLE int*2  handle to data file.
*******************************************************************
$INCLUDE:'BUFCOM'
$INCLUDE:'SETCOM'
$INCLUDE:'STATCOM'
      INTEGER*2 IOF(2),ICHAN(2),IPTR(8), MA(15),HANDLE
      CHARACTER*1 DR
      CHARACTER*3 EXT
      CHARACTER*8 N(8),U(8),XNAME,XUNITS, FN,STR1
      CHARACTER*30 TITLE
      CHARACTER*32 IMENU (15), BSTITL
      CHARACTER*60 BSPRMT
      REAL*4 YMIN(8),YMAX(8),YRANGE(8), YMXRNG (8),BASES (12)
      INTEGER*4 NPTS,ISTART,NPMAX,II,JJ,MBYTES,IP,IO,LBYTES,IIS,
     &            KCURS,NPTOT,MOVAV1, MOVAV2,NSMP

      DATA BSTITL /'AVERAGE OVER BASELENGTH...'/
      DATA BSPRMT /'AVERAGE OVER BASELENGTH:'/

      XNAME=CHID(10)
      XUNITS=UNITS(10)
      MAXBUF = MXBFSZ

*  Get baselength for moving average.

      IF (TSTTYP .EQ. 2) THEN
        MAXBS = MAXBUF / NCHRUT - NCHRUT
        IF (MAXBS .GT. NSRTOT) MAXBS = NSRTOT
        CALL GETLEN (AVEBAS, DXTRIM, MAXBS * DXTRIM, XUNITS, BSTITL,
     &           BSPRMT, IRET)
        IF (IRET .EQ. 1) RETURN

        MOVAV1 = AVEBAS / DXTRIM + 1
        MOVAV2 = MOVAV1 / 2 + 1
      ELSE
        MOVAV1 = 0
        MOVAV2 = 0
      END IF

*  Set limits and constants derived from the baselength.

      NPMAX = MAXBUF / NCHRUT - NCHRUT * MOVAV1 - NCHRUT
      IF (NPMAX .GT. NSRTOT - MOVAV1) NPMAX = NSRTOT - MOVAV1
      XSTART = MOVAV2 * DXTRIM
      NPTOT = NSRTOT - MOVAV1

*  Create title to pass to PLOT.
```

```
      CALL FNMAKE (DR, FN, EXT, PFILE, 1)
      TITLE = DR
      TITLE(2:2) = ':'
      TITLE(3:) = FN
      TITLE(13:) = 'AVE. BASE '
      L = 8
      CALL STRX (AVEBAS, STR1, L)
      TITLE (23:) = STR1(:L)
      IF (TSTTYP .EQ. 6) TITLE (13:) = 'BOUNCE'

C     PUT CHANNEL INFO INTO ARRAYS FOR PLTSEL CALL

      IF (LPROF) THEN
        N(ILIRI) = 'L. IRI'
        U(ILIRI) = UNITS(11)
        YMXRNG(ILIRI) = .03 * SCLFRI * SCLFDX / SCLFH
        IF (TSTTYP .EQ. 6) YMXRNG(ILIRI) = .25 / SCLFH
      END IF

      IF (RPROF) THEN
        N(IRIRI) = 'R. IRI'
        U(IRIRI) = UNITS(11)
        YMXRNG(IRIRI) = .03 * SCLFRI * SCLFDX / SCLFH
        IF (TSTTYP .EQ. 6) YMXRNG(IRIRI) = .25 / SCLFH
      END IF

      IF (RRUT) THEN
        N(IRR) = 'R. RUT'
        U(IRR) = UNITS(1)
        YMXRNG(IRR) = 2048. * GAIN (1)
      END IF

      IF (CRUT) THEN
        N(ICR) = 'C. RUT'
        U(ICR) = UNITS(1)
        YMXRNG(ICR) = 2048. * GAIN (1)
      END IF

      IF (LRUT) THEN
        N(ILR) = 'L. RUT'
        U(ILR) = UNITS(1)
        YMXRNG(ILR) = 2048. * GAIN (5)
      END IF

      IF (ICHV .NE. 0) THEN
        N(IVEL) = 'SPEED'
        U(IVEL) = UNITS(3)
        YMXRNG(IVEL) = 2048. * GAIN (3)
      END IF

      DO 5 II = 1, NCHRUT
        YRANGE(II)=YMXRNG(II)*.2
        CALL SCLUP (YMXRNG(II),X1,YMXRNG(II))
        CALL SCLUP (YRANGE(II),X1,YRANGE(II))
    5 CONTINUE

*   Set up default values for PLTSEL
```

```
      NCH = NCHRUT

      IOF (1) = 1
      IOF (2) = 2
      IF (XRANGE .LE. 0. .OR. XRANGE .GT. DXTRIM * NSRTOT) XRANGE = 10.
      KCURS = (XCURS - XSTART) / DXTRIM

C     SELECT CHANNEL(S) AND SCALE

      CALL PLTSEL(NCH, N, U, XNAME, XUNITS, DXTRIM, XMIN, XSTART,
     &              XRANGE, YRANGE, YMXRNG, NPTS, NPMAX, NPTOT,
     &              KCURS,IOF)

      IF(IOF (1) .EQ. 0 .AND. IOF (2) .EQ. 0) RETURN
      NCH = 1
      IF (IOF (2) .NE. 0) NCH = 2
      XCURS = KCURS * DXTRIM + XSTART

C     SET UP HALO
      CALL INITGR
      CALL SETIEEE(1)
      CALL FTSIZE (1,10)
      CALL FTCOLOR (1,0)
      CALL FTINIT

      DO 19 I=1,NCH
        J = IOF(I)
        YMAX (J) = YRANGE (J)
        YMIN (J) = -YRANGE (J)
        IF (J .EQ. IVEL .OR. J .EQ. ILIRI .OR. J .EQ. IRIRI)
     &                 YMIN (J) = 0.
   19 CONTINUE
      XMAX = XMIN + XRANGE

      NCHTOT = NCHRUT
      IACTIV = 1
      IUPDT = 2

   20 ISTART=(XMIN - XSTART) / DXTRIM
      IF(NPTS .GT. NPTOT-ISTART) NPTS=NPTOT-ISTART
      IIS = 0

C     READ IN DATA

      NSMP = NPTS + MOVAV1
      CALL RDTAPD (HANDLE, PCBUFR, 2, ISTART, NSMP, IERR)

      IF (TSTTYP .EQ. 2) THEN
        DO 40 ICH = 1, NCH
          J = IOF(ICH)
          JJ = J
          IF (J .EQ. ILIRI .OR. J .EQ. IRIRI) THEN
            M1 = MOVAV1 * NCHRUT
            SCLF = SCLFRI / (MOVAV1 * DXTRIM)
            DO 30 II = JJ, NCHRUT* NSMP + JJ, NCHRUT
   30       PCBUFR (II) = (PCBUFR (II + M1) - PCBUFR (II +
     &                        NCHRUT)) * SCLF
          ELSE
```

```
           CALL LOPASS (PCBUFR (J), NCHRUT, NSMP, MOVAV1, MOVAV2)
         END IF
   40  CONTINUE
       END IF

C      PLOT
       CALL PLOT (1,IACTIV,NCH,NPTS,IOF,IIS,NCHTOT,DXTRIM,XMIN,XMAX,
     &        XSTART,KCURS,YMIN,YMAX,N,U,XNAME,XUNITS,G,O,IUPDT,
     &        ISTART,NPTOT,NPMAX,TITLE)

       IF(IUPDT .EQ. 3)THEN
         XCURS = KCURS * DXTRIM + XSTART
         XRANGE = XMAX - XMIN
         CALL CLOSEGR
         RETURN
       END IF

       GOTO 20
       END
```

PLOTSEL.FOR

```
$TITLE:'PLTSEL SUBROUTINE'
$STORAGE:2
$NOFLOATCALLS

      SUBROUTINE PLTSEL (NCHAN, NAME, UNITS, XNAME, XUNITS, DX,
     &          XMIN, XSTART, XRANGE, YRANGE, YMXRNG, NPTS,
     &              NPMAX, NPTOT, KCURS, ICH)
***********************************************************************
*  Get plot settings from user.
*
*  --> NCHAN   int*2   number of channels.
*  --> NAME    char*8  array with names of each channel.
*  --> UNITS   char*8  array with units for each channel.
*  --> XNAME   char*8  name of variable plotted on x axis.  (time, etc.)
*  --> XUNITS  char*8  name of units for x axis.
*  --> DX      real*4  sample interval.
*  <-> XMIN    real*4  minimum limit of plotting range.
*  --> XSTART  real*4  x value at start of file (i=0).
*  <-> XRANGE  real*4  plotting range for x axis.
*  <-> YRANGE  real*4  array with plotting ranges for y axis.
*  --> YMXRNG  real*4  array with max allowable range for each channel.
*  <-- NPTS    int*4   number of points to plot.
*  --> NPMAX   int*4   maximum number of points that can be plotted.
*  --> NPTOT   int*4   maximum number of points in file.
*  <-> KCURS   int*4   position of cursor in file (0=1st point).
*  <-> ICH     int*2   array containing the 2 channels to be plotted.
*
      INTEGER*4 NPTS,NPMAX,NPTOT,KCURS
      INTEGER*2 ICH(*), MA(12)
      CHARACTER*8 NAME(*),UNITS(*),XUNITS,XNAME,STR1,STR2
      CHARACTER*32 IM(12),S(2),MENXR,MENSCL
      CHARACTER*60 PRMXR,PRMSCL
      REAL YRANGE (*),YMXRNG (*)

      DATA MENXR/'SELECT RANGE FOR X-AXIS'/
      DATA PRMXR/'RANGE OF X-AXIS COVERED IN ONE PLOT:'/
      DATA PRMSCL/'FULL SCALE:'/
      DATA MA/12*1/

C     GET X-AXIS RANGE.

      X1 = 10 * DX
      CALL SCLUP (X1, X, XLL)
      X1 = NPMAX * DX
      IF (NPMAX .GT. NPTOT) X1 = NPTOT * DX
      CALL SCLUP (X1, X, XUL)
      CALL GETLEN (XRANGE, XLL, XUL, XUNITS, MENXR, PRMXR, IRET)
      IF (IRET .EQ. 1) THEN
        ICH (1) = 0
        ICH (2) = 0
        RETURN
      END IF

      NPTS = XRANGE / DX + 2
```

```
*  Set initial cursor position (and the XMIN needed to include it).

      CALL CLRSCR
      L = 8
      CALL STRX (XSTART, STR1, L)
      L2 = 8
      XUL = (NPTOT - 1) * DX + XSTART
      CALL STRX (XUL,STR2,L2)

      CALL SETCUR (9, 0)
      WRITE(*,'(A,A,A,A,A,A\)')
     &      'THE PLOT CURSOR CAN BE SET ANYWHERE FROM ',STR1(:L),' TO ',
     &  STR2(:L2),' ',XUNITS

      CALL SETCUR(10,0)
      WRITE(*,'(A,A,A\)')'SET IT TO ',XNAME,' ='

      XCURS = KCURS * DX + XSTART
      IF (XCURS .LE. XSTART) XCURS = XSTART + DX
      IF (XCURS .GE. XUL) XCURS = XUL - DX

      CALL GETR(XCURS,XSTART, XUL ,10,22,9,'(F9.3\)',IRET)
      XMIN = AINT (XCURS * 2 / XRANGE) * XRANGE * .5
      IF (XCURS - XMIN .LT. XRANGE * .25) XMIN = XMIN - XRANGE * .5
      IF (XMIN .LT. XSTART) XMIN = XSTART + DX
      KCURS = (XCURS - XSTART) / DX

*  Select channels to plot.
*  If there is just one channel, don't bother the user.

      IF (NCHAN .EQ. 1) THEN
        ICH (1) = 1
        ICH (2) = 0
        L = 1
        GO TO 150

*  2 channels to choose from

      ELSE IF (NCHAN .EQ. 2) THEN
        IM(1) ='PLOT'
        DO 60 J = 1,2
   60   IM(J+1) = NAME(J)
        IM (4) = 'BOTH'
        IM (4) (6:) = NAME (3)
        IM(5) = 'CANCEL'
        IDEF = 3
        IF (ICH(1) .EQ. 1 .AND. ICH(2) .EQ. 0) IDEF = 1
        IF (ICH(1) .EQ. 2 .AND. ICH (2) .EQ. 0) IDEF = 2

        CALL MENU (IM, 5, MA, IDEF, IRET)
        IF (IRET .EQ. 4) THEN
          ICH (1) = 0
          ICH (2) = 0
          RETURN
        ELSE IF (IRET .EQ. 3) THEN
          ICH (1) = 1
          ICH (2) = 2
```

```
          L = 2
        ELSE
          L = 1
          ICH (1) = IRET
          ICH (2) = 0
        END IF

*   3 or more channels to choose from

      ELSE
        DO 50 I = 1, NCHAN
   50   IM (I + 1) = NAME (I)
        IM (1) = 'CHOOSE FIRST CHANNEL TO PLOT'
        IM (NCHAN + 2) = 'CANCEL'
          IF (ICH(1) .LT. 1 .OR. ICH (1) .GT. NCHAN) ICH (1) = 1
        CALL MENU (IM, NCHAN + 2, MA, ICH (1), IRET)
        IF (IRET .EQ. NCHAN + 1) THEN
          ICH (1) = 0
          ICH (2) = 0
          RETURN
        END IF
        ICH(1)=IRET
        L = 1

        IM (1) = 'CHOOSE SECOND CHANNEL TO PLOT'
        IM (NCHAN + 2) = 'JUST DO THE FIRST PLOT'
        IF (ICH(1) .EQ. ICH(2)) ICH(2) = ICH(1) + 1
        IF (ICH(2) .LT. 1 .OR. ICH(2) .GT. NCHAN) ICH(2) = 1
        IF (ICH(2) .EQ. ICH(1)) ICH(2) = 2
        CALL MENU (IM, NCHAN + 2, MA, ICH(2), IRET)
        IF (IRET .EQ. NCHAN + 1) THEN
          ICH (2) = 0
          L = 1
        ELSE
          ICH(2)=IRET
          L = 2
        END IF
      END IF

C     GET SCALES

150   CONTINUE

      DO 200 I=1,L
        ICHI = ICH(I)
        WRITE (MENSCL, 9020) NAME (ICHI)
9020    FORMAT('SELECT FULL SCALE FOR ',A8)

        X = YMXRNG (ICHI) * .2
        CALL SCLUP (X, X1, YRANGE (ICHI))
        X = YMXRNG(ICHI) * .005
        CALL SCLDWN (X,X1, YLL)
        CALL GETLEN (YRANGE(ICHI),YLL,YMXRNG(ICHI),UNITS(ICHI),
     &              MENSCL,PRMSCL,IRET)
        IF (IRET .EQ. 1) RETURN
200   CONTINUE

1000  RETURN
```

END

plotsubs.for

```
$TITLE:'PLOT SUBROUTINES'
$STORAGE:2
$NOFLOATCALLS

      SUBROUTINE GRCURS (ISTART, IPLT, KCURS, NPTS, IMAX, NPTOT,
     &             NPMAX,IUPDT, XMIN, XMAX, XSTART, DX, YMIN, YMAX,
     &             ICH)
*******************************************************************
*  Interpret cursor keys.  Move cursor and update plot parameters
*  if indicated.
*  --> ISTART int*4  offset (in file) to 1st point in plot.
*  <-> IPLT   int*2  number of active plot (1 or 2).
*  <-> KCURS  int*4  offset to present cursor position.
*  <-> NPTS   int*4  number of points on the screen.
*  --> IMAX   int*2  number of plots on screen (1 or 2).
*  --> NPTOT  int*4  number of points in data file.
*  --> NPMAX  int*4  max number of points that can be plotted.
*  <-- IUPDT  int*2  return cod.  0 =  I and KCURS updated; 1 = changed
*                    limits for one plot; 2 = changed limits for 2
*                    plots; 3=quit.
*  <-> XMIN   real*4 minimum x value.
*  <-> XMAX   real*4 maximum x value.
*  --> XSTART real*4 value of x at start of file (i=0).
*  --> DX     real*4 sample interval.
*  <-> YMIN   real*4 array with min y values for each channel in file.
*  <-> YMAX   real*4 array with max y values for each channel in file.
*  --> ICH    int*2  array with id no's of plotted channels.

      INTEGER*2 ICH(*)
      INTEGER*4 KK,NPTOT,NPTS, ISTART, NPMAX,KCURS
      REAL YMIN(*), YMAX(*)

      KKINDEX=NPTS / 20 + 1
      KK = KCURS - ISTART

C     WAIT FOR A KEY
10    CALL KCLEAR
20    J=IGKEY()
      IF(J.EQ. 0) GOTO 20
      IUPDT=0

C     PRINT SCREEN IF KEY WAS 'P'
      IF (J .EQ. 80 .OR. J .EQ. 112) THEN
        CALL SETGPR(1)
        CALL GPRINT
        GOTO 10
      ELSE IF (J .EQ. 13) THEN
        IPLT = IMAX - IPLT + 1

*  Zoom keys.  after + or -, wait for next key(s) to finish.
*  Can have 1 or 2 zoom keys followed by x or y.

      ELSE IF (J .EQ. 43 .OR. J .EQ. 45) THEN
```

```
        IF (J .EQ. 45) THEN
          ZOOM = 1.5
   25     CALL KCLEAR
   30       J = IGKEY()
          IF (J .EQ. 0) THEN
            GO TO 30
          ELSE IF (J .EQ. 45) THEN
            ZOOM = 4.
            GO TO 25
          ELSE IF (J .EQ. 88 .OR. J .EQ. 120) THEN
            GO TO 100
          ELSE IF (J .EQ. 89 .OR. J .EQ. 121) THEN
            GO TO 200
          END IF
        ELSE IF (J .EQ. 43) THEN
          ZOOM = 1/3.
   35     CALL KCLEAR
   40       J = IGKEY()
          IF (J .EQ. 0) THEN
            GO TO 40
          ELSE IF (J .EQ. 43) THEN
            ZOOM = 1/6.
            GO TO 35
          ELSE IF (J .EQ. 88 .OR. J .EQ. 120) THEN
            GO TO 100
          ELSE IF (J .EQ. 89 .OR. J .EQ. 121) THEN
            GO TO 200
          END IF
        END IF

C     REST ARE ALL CURSOR KEYS
      ELSE IF( J .GT. 0) THEN
       GOTO 10

C     CHECK FOR UP ARROW
      ELSE IF( J .EQ. -72)THEN
        YR = (YMAX(ICH(IPLT)) - YMIN(ICH(IPLT))) * .25
        YMAX(ICH(IPLT)) = YMAX(ICH(IPLT)) + YR
        YMIN(ICH(IPLT)) = YMIN(ICH(IPLT)) + YR
        IUPDT = 1

C     CHECK FOR DOWN ARROW
      ELSE IF(J .EQ. -80)THEN
        YR = (YMAX(ICH(IPLT)) - YMIN(ICH(IPLT))) * .25
        YMAX(ICH(IPLT)) = YMAX(ICH(IPLT)) - YR
        YMIN(ICH(IPLT)) = YMIN(ICH(IPLT)) - YR
        IUPDT = 1

C     CHECK FOR RIGHT ARROW
      ELSE IF(J .EQ. -77)THEN
       IF (KK .LT. NPTS- 1)KK = KK + 1

C     CHECK FOR LEFT ARROW
      ELSE IF(J .EQ. -75)THEN
       IF (KK .GT. 0) KK = KK - 1

C     CHECK FOR CONTROL RIGHT ARROW
      ELSE IF ( J .EQ. -116)THEN
```

```
       IF(KK+KKINDEX .LE. NPTS)KK = KK + KKINDEX

C     CHECK FOR CONTROL LEFT ARROW
      ELSE IF ( J .EQ. -115)THEN
       IF(KK-KKINDEX .GE. 0) KK = KK - KKINDEX

C     PAGE UP (FULL OR HALF)
      ELSE IF(J .EQ. -73 .OR. J .EQ. -132) THEN
        XR = XMAX - XMIN
        IF (J .EQ. -73) THEN
          IF(ISTART+NPTS .GT. NPTOT) GOTO 10
          XMIN = XMIN + XR
        ELSE
          IF(ISTART+NPTS/2 .GE. NPTOT) GOTO 10
          XMIN = XMIN + XR * .5
        END IF
        NPTS = XR / DX + 1
        XMAX = XMIN + XR
        ISTART = (XMIN - XSTART) / DX
        IF (ISTART + NPTS .GT. NPTOT) NPTS = NPTOT - ISTART
        IUPDT = 2

C     PAGE DOWN
      ELSE IF (J .EQ. -81 .OR. J .EQ. -118) THEN
        XR = XMAX - XMIN
        NPTS = XR / DX
        IF (J .EQ. -81) THEN
          IF(ISTART .LT. NPTS)THEN
            XMIN=XSTART
          ELSE
            XMIN = XMIN - XR
          END IF
        ELSE
          IF (ISTART .LT. NPTS / 2) THEN
            XMIN = XSTART
          ELSE
            XMIN = XMIN - XR * .5
          END IF
        ENDIF
        XMAX = XMIN + XR
        ISTART = (XMIN - XSTART) / DX
        IF (ISTART + NPTS .GT. NPTOT) NPTS = NPTOT - ISTART
        IUPDT = 2
*  Quit.

      ELSE IF (J .EQ. -79) THEN
        IUPDT = 3

*  Nothing valid, so wait for next key.

      ELSE
        GO TO 10
      END IF
      KCURS = ISTART + KK
      IF (KCURS .GT. NPTOT) KCURS = ISTART + 1
      RETURN

*  Zoom in/out of x.
```

```
  100 CONTINUE
      KCURS = ISTART + KK
      IF (KCURS .GT. NPTOT) KCURS = ISTART + 1
      CALL SCLUP (ZOOM, X1, ZOOM)
      XR = XMAX - XMIN
      XR = XR * ZOOM
      XMIN = AINT ((DX * KCURS) / XR) * XR + XSTART
      ISTART = (XMIN - XSTART) / DX
      XMAX = XMIN + XR
      NPTS = XR / DX + 1
      IF (ISTART + NPTS .GT. NPTOT) NPTS = NPTOT - ISTART
      IF (NPTS .GT. NPMAX) NPTS = NPMAX
      IUPDT = 2
      RETURN

*  Change scale factor of y axis.
  200 CALL SCLUP (ZOOM, X1, ZOOM)
      IF (YMAX(ICH(IPLT)) * YMIN(ICH(IPLT)) .EQ. 0.) THEN
        YMAX(ICH(IPLT)) = YMAX (ICH(IPLT)) * ZOOM
        YMIN(ICH(IPLT)) = YMIN(ICH(IPLT)) * ZOOM
      ELSE
        YM = (YMAX(ICH(IPLT)) + YMIN(ICH(IPLT))) * .5
        YR = (YMAX(ICH(IPLT)) - YMIN(ICH(IPLT))) * .5 * ZOOM
        YMAX(ICH(IPLT)) = YM + YR
        YMIN(ICH(IPLT)) = YM - YR
      END IF
      IUPDT = 1
      KCURS = KK + ISTART
      IF (KCURS .GT. NPTOT) KCURS = ISTART + 1
      RETURN

      END
$PAGE
```

```
      SUBROUTINE SCLUP(X,XNORM,XUP)
*******************************************************************
C     THIS SUBROUTINE SCALES A VARIABLE UP TO THE NEXT
C     MULTIPLE OF 1 , 2  OR 5

      MAG=ALOG10(X)
      IF(X .LT. 1) MAG=MAG-1
      XNORM=X* .1 ** MAG
      XUP=2.
      IF(XNORM .GT. 2.02 .AND. XNORM .LE. 5.05)XUP=5.
      IF ( XNORM .GT. 5.05) XUP=10.
      IF ( XNORM .LT. 1.01)XUP=1.
      XUP=XUP*10.0**MAG
      RETURN
      END
$PAGE
```

```
      SUBROUTINE SCLDWN (X,XNORM,XDOWN)
*****************************************************************
C     THIS SUBROUTINE SCALES A VARIABLE DOWN TO THE NEXT
C     MULTIPLE OF 1 , 2  OR 5

      MAG=ALOG10(X)
      IF(X .LT. 1) MAG=MAG-1
      XNORM=X* .1 ** MAG
      XDOWN=2.
      IF(XNORM .GT. 1.98 .AND. XNORM .LE. 4.95) XDOWN=2.
      IF ( XNORM .GT. 5.05) XDOWN=5.
      IF ( XNORM .GT. 9.9)XDOWN=10.
      XDOWN=XDOWN*10.0**MAG
      RETURN
      END

$PAGE
```

```
      SUBROUTINE TIKSET (XMIN, XMAX, TICK, TMIN, TMAX, NTICK)
*********************************************************************
*  Determine first and last tick marks in a given range.
*
*  --> XMIN    real*4 minimum limit in range (eng. units).
*  --> XMAX    real*4 maximum linit to range (eng. units).
*  --> TICK    real*4 tick interval (eng. units).
*  <-- TMIN    real*4 first tick interval within range.
*  <-- TMAX    real*4 last tick interval within range.
*  <-- NTICK   int    number of ticks within range.
*********************************************************************
      TMIN = INT ( XMIN / TICK - .01) * TICK
        TMAX = INT (XMAX / TICK + .01) * TICK
        IF (TMIN .LT. XMIN - .01 * TICK) TMIN = TMIN + TICK
        IF (TMAX .GT. XMAX + .01 * TICK) TMAX = TMAX - TICK
        NTICK = (TMAX - TMIN) / TICK + 1.5
        RETURN
        END
*********************************************************************
      SUBROUTINE LABEL (X, STRING, L)
*********************************************************************
*  This subroutine converts a real number into a string for Halo.
*
*  --> X      real*4  number to be converted
*  <-- STRING char*10 string representation of X, with beginning and
*                     ending \ characters for Halo.
*  <-- L      int*2   number of characters in STRING. (Not counting
*                     beginning and ending \'s.)
*********************************************************************
      CHARACTER*10 STRING
      CHARACTER*13 KEYBUF
      INTEGER*2 L
      L = 8
      STRING(1:1) = '\'
      CALL STRX (X, KEYBUF, L)
      STRING (2:) = KEYBUF (:L)
      STRING (2+L:) = '\'
      RETURN
      END
```

PRFCMP.FOR

```
$TITLE:'SUBROUTINE PRFCMP'
$NOFLOATCALLS
$STORAGE:2
      SUBROUTINE PRFCMP (HANDLE)
**********************************************************************
*  This subroutine does the basic signal processing, converting a
*  data file from an integer*2 representation of raw data into 3
*  inter-leaved real*4 representations of slope profile, rut depth,
*  and elevation profile.
**********************************************************************
*
*  Logical keys--true if output channel will be created [SETUP]:
*     RPROF, LPROF, LRUT, CRUT, RRUT
*     TSTTYP = 3 for normal test, 5 for bounce test.
*            = 7 if damaged during processing.
*
*  Channel ID numbers for 8 raw data channels [SETUP]:
*     ICHH1, ICHA1, ICHV, ICHA2, ICHH2, ICHH3, ICHH4, ICHH5
*
*  Channel ID numbers for 2 data channels in profile files [SETUP]:
*     ILPRF, IRPRF
*
*  Channel ID numbers for 6 data channels in rut file [SETUP]:
*     ILIRI, IRIRI, IVEL, ILR, ICR, IRR
*
*  Buffers:
*     MAXBUF = number of available full-words for all buffers,
*              including the raw input data and all output [SETUP].
*     PCBUFI = integer*2 buffer available from common in
*              IBM PC [PCBUF].
*     PCBUFR = real*4 buffer equivalenced to PCI2 [PCBUF].
*
*  Number of channels in buffers and files [SETUP]:
*     NCHPRF =  no. of channels in profile files (PRFBUF and ELVBUF).
*     NCHRUT =  no. of channels in compressed Rut file.
*     NCHRAW =  no. of channels of raw input.
*
*  Number of full-words (4-bytes) in buffers [LOCAL]:
*     NBUFFW = no. of full-words between buffer starts on tape.
*     NELVFW = no. of full-words in elevation buffer.
*     NPRFFW = no. of full-words in slope profile buffer.
*     NRAWFW = no. of full-words in input buffer.
*     NRUTFW = no. of full-words in RUT buffer.
*
*  Number of samples in buffers
*  (Note: "Npoints" = "Nsamp" x "Nchans") [LOCAL]:
*     NFSAMP = no. of samples used to initialize filter.
*     NPSAMP = no. of samples/buffer for slope profile. Input
*              buffer includes 1 extra point.
*     NRSAMP = no. of samples/buffer for rut and elevation data.
*
*  Number of samples in test
*  (Note: "Npoints" = "Nsamp" x "Nchans") [SETUP]:
*     NBUFS  = no. of buffers in entire test.
```

```
*     NSAMP  = no. of raw data samples in entire test.
*     NSPTOT = no. of profile samples in profile file.
*     NSRTOT = no. of rut samples in compressed file.
*
*  Counters [LOCAL]:
*     LSTBUF = number of previous buffer read from tape.
*
*  Computation constants:
*     COFINT = coefficient used in profile computation [LOCAL].
*              COFINT = 1 - DELTAX/LNGWAV
*     DELTAX = sample interval for raw data [SETUP].
*     LNGWAV = time-constant for high-pass filter built into the
*              profile computation.  Actually, LNGWAV has units of
*              length (meters) rather than time [SETUP].
*     TRIM   = decimation ratio for rut data.  (Every TRIM-th point
*              is kept after decimation.)  TRIM must be an even
*              no [SETUP].
*
*   Gains and scale factors used in computations:
*     GAIN   = array of gains for raw data. [SETUP].
*     GAINAL = accel/speed gain used in computing L. profile [LOCAL].
*     GAINAR = accel/speed gain used in computing R. profile [LOCAL].
*     GAINHL, CGHL = height gains used in computing L. profile [LOCAL].
*     GAINHR, CGHR = height gains used in computing R. profile [LOCAL].
*     SCLFA  = scale factor used convert acceleration to m/s/s [SETUP].
*     SCLFDX = scale factor used convert DELTAX to m [SETUP].
*     SCLFH  = scale factor used convert height to m [SETUP].
*     SCLFV  = scale factor used convert test speed to m/s [SETUP].
*
*   Biases [LOCAL]:
*     BACC1  = bias in accelerometer #1 (integer*2).
*     BACC2  = bias in accelerometer #2 (integer*2).
*     BLPRF  = bias in left slope profile signal (real).
*     BRPRF  = bias in right slope profile signal (real).
*     BVEL   = bias in velocity signal (integer*2).
*
*   Integrals [LOCAL]:
*     LAINT  = integral of left accelerometer.
*     RAINT  = integral of right accelerometer.
*     LELEV  = elevation of left profile at end of buffer.
*     RELEV  = elevation of right profile at end of buffer.
*
*  Variables used in quarter-car simulation [LOCAL]:
*     XL1, XL2, XL3, XL4 = state variables for 1/4 car on left.
*     XR1, XR2, XR3, XR4 = state variables for 1/4 car on right.
*     LROUGH = roughness of left profile at end of buffer.
*     RROUGH = roughness of right profile at end of buffer.
*
*  Special functons and subroutines called by prfcmp:
*     AVEVEL -- smooths and decimates signal.  Input is integer*2
*               array; output is real*4 array.
*     DEBIAS -- remove bias from signal in 2-D real*4 array.
*     IAVE   -- finds average of signal in 2-D integer*2 array.
*     PRFELV -- compute elevation profile from slope profile.
*     PRFIRI -- compute IRI roughness from slope profile.
*     RAVE   -- finds average of signal in 2-D real*4 array.
*               array; output is real*4 array.
*     RDTAPE -- read binary data from tape file.
```

```
*     WRTAPE -- write binary data to tape file.
*
***********************************************************************
$INCLUDE:'SETCOM'
$INCLUDE:'BUFCOM'
***********************************************************************
      INTEGER*2 BACC1, BACC2, BVEL, IAVE, IERR, HANDLE
      INTEGER*4 I, IREC, IAL, IAR, IHL, IHR, IPL, IPR, IV, LSTBUF,
     &          NFSAMP, NRAWFW, OFFSET, NBYTES, I11
      REAL*4 GAINAL, GAINAR, CGHL, CGHR, LAINT, RAINT, GAINHR,
     &       GAINHL, VEL, LELEV, RELEV, LROUGH, RROUGH

*   Create bogus deltax if this is a bounce test.

      IF (TSTTYP .EQ. 5) THEN
        DT = IDIV * .4190477E-06
        V = (3800. * GAIN (3) - ZDATA (3)) * SCLFV
        DELTAX = DT * V / SCLFDX
        LNGWAV = 1. * DELTAX / DT
        DXTRIM = DX * TRIM
        CALL SETSTM
      END IF
*
*   Set the number of samples contained in the PC buffer.  Choose
*   sizes to maximize the amount of data processed in each buffer.
*   First calculate sizes assuming the whole test fits in one buffer.
*   Then check this assumption, and set up for multiple buffers if
*   necessary.
*
      NRSAMP = (NSAMP - 1) / TRIM
      NRUTFW = NRSAMP * NCHRUT
      NRAWFW = NCHRAW * NSAMP / 2 + 2 + .5
      MAXBUF = NRAWFW + NRUTFW
      NPSAMP = NRSAMP * TRIM
      NBUFS = 1
*
      IF (MAXBUF .GT. MXBFSZ) THEN
        MAXBUF = MXBFSZ
        NRSAMP = (MAXBUF - 2 - NCHRAW) / (NCHRUT + TRIM * NCHRAW / 2)
        NPSAMP = TRIM * NRSAMP
        NBUFS = (NSAMP - 1) / NPSAMP
        IF (MOD (NPSAMP, NSAMP - 1) .NE. 0) NBUFS = NBUFS + 1
      END IF

*   Set the long-wave cutoff as a funtion of the minimun test speed.

      IF (TSTTYP .EQ. 3) THEN
        LNGWAV = VELMIN * 7. * SCLFV / SCLFDX
        NFSAMP = 6 * LNGWAV / DELTAX
      ELSE
        NFSAMP = NPSAMP - 1
      END IF
      IF (NFSAMP .GT. NPSAMP - 1) NFSAMP = NPSAMP - 1
*
*   Calculate number of buffers and total samples for test.
*   We lose the last sample (to differientiate the height signals).
*
      NSRTOT = (NSAMP - 1) / TRIM
```

```
      NSPTOT = NSRTOT * TRIM
*
*  Set the number of words in the various portions of buffers.
*
      NRUTFW = NCHRUT * NRSAMP
      NPRFFW = NCHPRF * NPSAMP
      NELVFW = NCHPRF * NRSAMP
      NRAWFW = (NCHRAW * (NPSAMP + 1) + 1) / 2 + 2
      NBUFFW = NCHRAW * NPSAMP / 2
*
*  Initialize counters and integrals.
*
      LELEV = 0
      RELEV = 0
      LAINT = 0
      RAINT = 0
      LSTBUF = 0
*******************************************************************
*  Update status report on screen, then read next buffer.
*
   15 CONTINUE
      CALL CLRLIN (20)
      CALL SETCUR (20,0)
      WRITE (*,'(A,I2,A,I2\)') 'NOW PERFORMING FIRST PASS ON BUFFER #',
     &       LSTBUF + 1,' OF',NBUFS
      CALL SETCUR (21,0)
      WRITE (*,'(A\)') 'READING IN DATA...                          '
*
      OFFSET = LSTBUF * NBUFFW * 4
      NBYTES = NRAWFW * 4
      CALL RDTAPE (HANDLE, PCBUFI, OFFSET, NBYTES, IERR)
*
      CALL SETCUR (21,0)
      WRITE (*,'(A\)') 'CONDITIONING SIGNALS...     '
*
*  Modify number of samples if this is the last buffer.
*
      IF (LSTBUF .GE. NBUFS - 1) THEN
        NRSAMP = (NSAMP - 1 - NPSAMP * LSTBUF) / TRIM
        NPSAMP = TRIM * NRSAMP
      END IF
*
*  Average and decimate speed signal.
*
      IF (ICHV .GT. 0)
     &  CALL AVEVEL (PCBUFI (ICHV), NCHRAW, NPSAMP, PCBUFR (IVEL +
     &        NRAWFW), NCHRUT, TRIM, GAIN (3), ZDATA (3))
*
*  Compute up to three rut-depth signals.
*
      CALL SETCUR (21,0)
      WRITE (*,'(A\)') 'COMPUTING RUT DEPTH...      '
      IF (LRUT)
     &     CALL RUTCMP (PCBUFI (ICHH4), PCBUFI (ICHH2), PCBUFI
     &          (ICHH3), NCHRAW, NPSAMP, PCBUFR (NRAWFW + ILR),
     &          NCHRUT, TRIM, GAIN (7), GAIN (5), GAIN (6),
     &          ZDATA (7), ZDATA (5), ZDATA (6),
     &          H4LAT, H2LAT)
```

```
      IF (CRUT)
     &      CALL RUTCMP (PCBUFI (ICHH2), PCBUFI (ICHH3), PCBUFI
     &           (ICHH1), NCHRAW, NPSAMP, PCBUFR (NRAWFW + ICR),
     &           NCHRUT, TRIM, -GAIN (5), -GAIN (6), -GAIN (1),
     &           -ZDATA (5), -ZDATA (6), -ZDATA (1),
     &           H2LAT, H1LAT)
      IF (RRUT)
     &      CALL RUTCMP (PCBUFI (ICHH3), PCBUFI (ICHH1), PCBUFI
     &           (ICHH5), NCHRAW, NPSAMP, PCBUFR (NRAWFW + IRR),
     &           NCHRUT, TRIM, GAIN (6), GAIN (1), GAIN (8),
     &           ZDATA (6), ZDATA (1), ZDATA (8),
     &           H1LAT, H5LAT)
*
*  Compute bias in accelerometer signal(s) if this is the first
*  buffer.  Also, copy gains into scaler variables.  The slope profile
*  will have units: H/L, where H are the units of the height sensor
*  and L are the units of the sample interval DELTAX.  As a result,
*  all subsequent elevation signals will have the same units as the
*  height sensors.
*
      CALL SETCUR (21,0)
      WRITE (*,'(A\)') 'COMPUTING SLOPE PROFILE...    '
*
      IF (ICHV .NE. 0) BVEL = ZDATA (3) / GAIN (3)
      COFINT = 1. - DELTAX / LNGWAV
      IF (LPROF .AND. (LSTBUF .EQ. 0)) THEN
        BACC2 = IAVE (PCBUFI (ICHA2), NCHRAW, NFSAMP)
        GAINAL = -GAIN (4) * SCLFA * DELTAX / (GAIN (3) *
     &            SCLFV) ** 2 / SCLFH * SCLFDX ** 2
        GAINHL = -GAIN (5) / DELTAX
        CGHL = COFINT * GAINHL
      END IF
*
      IF (RPROF .AND. (LSTBUF .EQ. 0)) THEN
        BACC1 = IAVE (PCBUFI (ICHA1), NCHRAW, NFSAMP)
        GAINAR = -GAIN (2) * SCLFA * DELTAX / (GAIN (3) *
     &            SCLFV) ** 2 / SCLFH * SCLFDX ** 2
        GAINHR = -GAIN (1) / DELTAX
        CGHR = COFINT * GAINHR
      END IF
*
*  Initialize pointers for profile computation.
*
      IAL = ICHA2
      IHL = ICHH2
      IAR = ICHA1
      IHR = ICHH1
      IV = ICHV
      IPL = ILPRF
      IPR = IRPRF
*
*  Compute slope profile for case of left profile only.
*
      IF (LPROF .AND. .NOT. RPROF) THEN
        DO 50 I = 1, NPSAMP
          VEL = PCBUFI (IV) - BVEL
          LAINT = COFINT * LAINT + GAINAL * (PCBUFI (IAL) - BACC2) /
     &            VEL ** 2
```

```
        PCBUFR (IPL) = LAINT + CGHL * PCBUFI (IHL + NCHRAW) -
     &          GAINHL * PCBUFI (IHL)
        IAL = IAL + NCHRAW
        IHL = IHL + NCHRAW
        IV = IV + NCHRAW
        IPL = IPL + NCHPRF
   50 CONTINUE
      END IF
*
*  Compute slope profile for case of right profile only.
*
      IF (RPROF .AND. .NOT. LPROF) THEN
        DO 60 I = 1, NPSAMP
          VEL = PCBUFI (IV) - BVEL
          RAINT = COFINT * RAINT + GAINAR * (PCBUFI (IAR) - BACC1) /
     &            VEL ** 2
          PCBUFR (IPR) = RAINT + CGHR * PCBUFI (IHR + NCHRAW) -
     &            GAINHR * PCBUFI (IHR)
          IAR = IAR + NCHRAW
          IHR = IHR + NCHRAW
          IV = IV + NCHRAW
          IPR = IPR + NCHPRF
   60   CONTINUE
      END IF
*
*  Compute slope profile for case of both profiles.
*
   69 IF (RPROF .AND. LPROF) THEN
        DO 70 I = 1, NPSAMP
          VEL = PCBUFI (IV) - BVEL
          RAINT = COFINT * RAINT + GAINAR * (PCBUFI (IAR) - BACC1) /
     &            VEL ** 2
          PCBUFR (IPR) = RAINT + CGHR * PCBUFI (IHR + NCHRAW)
     &           - GAINHR * PCBUFI (IHR)
          LAINT = COFINT * LAINT + GAINAL * (PCBUFI (IAL) - BACC2) /
     &            VEL ** 2
          PCBUFR (IPL) = LAINT + CGHL * PCBUFI (IHL + NCHRAW)
     &           - GAINHL * PCBUFI (IHL)
          IAL = IAL + NCHRAW
          IHL = IHL + NCHRAW
          IAR = IAR + NCHRAW
          IHR = IHR + NCHRAW
          IV = IV + NCHRAW
          IPL = IPL + NCHPRF
          IPR = IPR + NCHPRF
   70   CONTINUE
      END IF
*
*  Compute and correct bias in slope profile(s) if this is
*  the first buffer.  Also initialize quarter-car here.
*
      IF (LPROF .AND. (LSTBUF .EQ. 0)) THEN
        BLPRF = RAVE (PCBUFR (ILPRF), NCHPRF, NFSAMP)
        CALL DEBIAS (PCBUFR (ILPRF), NCHPRF, NPSAMP, BLPRF)
        LAINT = LAINT - BLPRF
*
        I11 = 11. / DELTAX
        XL1 = RAVE (PCBUFR (ILPRF), NCHPRF, I11)
```

```
        XL2 = 0
        XL3 = XL1
        XL4 = 0
        LROUGH = 0
      END IF
*
      IF (RPROF .AND. (LSTBUF .EQ. 0)) THEN
        BRPRF = RAVE (PCBUFR (IRPRF), NCHPRF, NFSAMP)
        CALL DEBIAS (PCBUFR (IRPRF), NCHPRF, NPSAMP, BRPRF)
        RAINT = RAINT - BRPRF
*
        I11 = 11. / DELTAX
        XR1 = RAVE (PCBUFR (IRPRF), NCHPRF, I11)
        XR2 = 0
        XR3 = XR1
        XR4 = 0
        RROUGH = 0
      END IF

*  Before writing any data to tape, set TSTTYP to 7 so that if
*  something goes wrong the file will be identified as unrecoverable.

      IDUMMY = TSTTYP
      TSTTYP = 7
      CALL UPDSET (HANDLE)
      TSTTYP = IDUMMY
*
*  Write profile buffer to tape.  Do this before computing IRI,
*  in case the IRI analysis eventually includes a moving average.
*
      IF (RPROF .OR. LPROF) THEN
        CALL SETCUR (21,0)
        WRITE (*,'(A\)') 'WRITING SLOPE PROFILE...     '
        OFFSET = LSTBUF * NBUFFW * 4
        NBYTES = NPRFFW * 4
        CALL WRTAPE (HANDLE, PCBUFR, OFFSET, NBYTES, IERR)
        CALL SETCUR (21,0)
        WRITE (*,'(A\)') 'CALCULATING IRI ROUGHNESS...     '
      END IF
*
*  Compute IRI roughness for profiles.
*
      IF (LPROF) CALL PRFIRI (PCBUFR (ILPRF), PCBUFR (ILIRI + NRAWFW),
     &             XL1, XL2, XL3, XL4, LROUGH)
      IF (RPROF) CALL PRFIRI (PCBUFR (IRPRF), PCBUFR (IRIRI + NRAWFW),
     &             XR1, XR2, XR3, XR4, RROUGH)
*
*  Write rut-depth buffer to tape.
*
      CALL SETCUR (21,0)
      WRITE (*,'(A\)') 'WRITING RUT DEPTH AND ROUGHNESS...     '
*
      OFFSET = (LSTBUF * NBUFFW + NPRFFW) * 4
      NBYTES = NRUTFW * 4
      CALL WRTAPE (HANDLE, PCBUFR (NRAWFW + 1), OFFSET, NBYTES, IERR)
*
*  Go back and read some more, unless this was the last buffer.
*
```

```
      LSTBUF = LSTBUF + 1
      IF (LSTBUF .LT. NBUFS) GO TO 15
*******************************************************************
*  Have now finished the first pass.  Make a second pass, to
*  integrate backwards and calculate the elevation benchmarks.
*
      IF (LPROF .OR. RPROF) THEN
*
*  Read slope profile from "tape."
*
        DO 100 LSTBUF = NBUFS - 1, 0, -1
          CALL SETCUR (20,0)
          WRITE (*,'(A,I2,A\)') 'SECOND PASS FOR BUFFER #',
     &                LSTBUF + 1,'                           '
          CALL SETCUR (21,0)
          WRITE (*,'(A\)') 'READING...                                  '
          OFFSET = LSTBUF * NBUFFW * 4
          NBYTES = NPRFFW * 4
          CALL RDTAPE (HANDLE, PCBUFR, OFFSET, NBYTES, IERR)
          CALL SETCUR (21,0)
          WRITE (*,'(A\)') 'COMPUTING ELEVATION...'
*
*  Compute elevation data.
*
          IF (LPROF) CALL PRFELV (PCBUFR (ILPRF), NCHPRF, NPSAMP,
     &       PCBUFR (NRAWFW + ILPRF), NCHPRF, TRIM, DELTAX, COFINT,
     &       LELEV)
          IF (RPROF) CALL PRFELV (PCBUFR (IRPRF), NCHPRF, NPSAMP,
     &       PCBUFR (NRAWFW + IRPRF), NCHPRF, TRIM, DELTAX, COFINT,
     &       RELEV)
*
*  Re-set NPSAMP for next call to PRFELV.
*
          NRSAMP = (MAXBUF - NCHRAW) / (NCHRAW * TRIM / 2 + NCHRUT)
          NPSAMP = NRSAMP * TRIM
*
*  Write elevation data to tape.
*
          CALL SETCUR (21,0)
          WRITE (*,'(A\)') 'WRITING...              '
          OFFSET = (LSTBUF * NBUFFW + NPRFFW + NRUTFW) * 4
          NBYTES = 4 * NELVFW
          CALL WRTAPE (HANDLE, PCBUFR (NRAWFW + 1), OFFSET, NBYTES,
     &                IERR)
  100   CONTINUE
      END IF
*
*  Change TSTTYP to indicate that it's not raw data any more.
*
      IF (TSTTYP .EQ. 3) TSTTYP = 2
      IF (TSTTYP .EQ. 5) THEN
        TSTTYP = 6
        LNGWAV = 1.
        DELTAX = DT
        DXTRIM = DELTAX * TRIM
      END IF
      CALL UPDSET (HANDLE)
      RETURN
```

END

PRFIRI.FOR

```
$TITLE:'SUBROUTINE PRFIRI'
$NOFLOATCALLS
$STORAGE:2

      SUBROUTINE PRFIRI (BUF1, BUF2, X1, X2, X3, X4, ROUGH)
*******************************************************************
*  This subroutine filters a slope profile signal using the IRI
*  quarter-car simulation.  The accumulated IRI roughness is
*  compressed and stored in a separate array.  The IRI coefficients
*  and the sizes of the arays are obtained from COMMON.  This
*  subroutine will probably be enhanced to smooth the
*  slope profiles, so it should not be called until the profiles are
*  stored on tape.
*
*  --> BUF1   real*4     2-D input array with profile data.  Ch-1 is
*                        processed.
*  <-- BUF2   real*4     2-D output array (with rut stuff also.)  Ch-1
*                        is replaced.
*  <-> X1-X4  real*4     vehicle response variables, updated every
*                        step.
*  <-> ROUGH  real*4     accumulated roughness, updated every step.
*******************************************************************
$INCLUDE:'SETCOM'
*
$LARGE: BUF1, BUF2
      INTEGER*4 I, I1, I2, J
      REAL*4 BUF1 (*), BUF2 (*), X1, X2, X3, X4,
     &       X1N, X2N, X3N, X4N, S11, S12, S13, S14, S21, S22, S23,
     &       S24, S31, S32, S33, S34, S41, S42, S43, S44, P1, P2, P3,
     &       P4, ROUGH
      EQUIVALENCE
     &  (STM (1, 1), S11), (STM (2, 1), S21), (STM (3, 1), S31),
     &  (STM (4, 1), S41), (STM (1, 2), S12), (STM (2, 2), S22),
     &  (STM (3, 2), S32), (STM (4, 2), S42), (STM (1, 3), S13),
     &  (STM (2, 3), S23), (STM (3, 3), S33), (STM (4, 3), S43)
      EQUIVALENCE
     &  (STM (1, 4), S14), (STM (2, 4), S24), (STM (3, 4), S34),
     &  (STM (4, 4), S44), (PRM (1), P1), (PRM (2), P2),
     &  (PRM (3), P3), (PRM (4), P4)
*
*  <No smoothing for now...>
*
*     CALL SMOOTHERUPPER...
*
*  Simulate vehicle response
*
      I1 = 1
      I2 = 1
      DO 40 I = 1, NRSAMP
        DO 30 J = 1, TRIM
          P = BUF1 (I1)
          X1N = X1 * S11 + X2 * S12 + X3 * S13 + X4 * S14 + P1 * P
          X2N = X1 * S21 + X2 * S22 + X3 * S23 + X4 * S24 + P2 * P
          X3N = X1 * S31 + X2 * S32 + X3 * S33 + X4 * S34 + P3 * P
```

```
      X4N = X1 * S41 + X2 * S42 + X3 * S43 + X4 * S44 + P4 * P
      X1 = X1N
      X2 = X2N
      X3 = X3N
      X4 = X4N
      ROUGH = ROUGH + DELTAX * ABS (X1 - X3)
30    I1 = I1 + NCHPRF
      BUF2 (I2) = ROUGH
40    I2 = I2 + NCHRUT
      RETURN
      END
```

PROCESS.FOR

```
$TITLE:'PROCESS'
$STORAGE:2
$NOFLOATCALLS

      SUBROUTINE PROCESS
************************************************************************
*
*  This subroutine generates the menu for viewing data and calls the
*  appropriate subroutines based on the items selected from that menu.

$INCLUDE:'STATCOM'
$INCLUDE:'HANDLES'
$INCLUDE:'SETCOM'
      CHARACTER*32 IMENU(20)
      INTEGER*2 MA(20),JUNK(1700)
      INTEGER*4 LSAT1
      CHARACTER*1 DR,PF(16)
      LOGICAL OPENFL, QNDPLT
      EQUIVALENCE(PFILE,PF)
      DATA OPENFL /.FALSE./

*     SAVE SETUP ARRAY
      CALL WRTSET

      MI=14
      DO 10 I=1,MI
10    MA(I)=0
      MA(1)=1
      MA(2)=1
      MA(4)=1
      MA(MI) = 1

      IMENU(1) ='VIEW AND PROCESS DATA'
      IMENU(2) ='OPEN TEST FILE'
      IMENU(3) ='OPEN BOUNCE FILE'
      IMENU(4) ='CHECK RAW DATA'
      IMENU(5) ='PRE-PROCESS FILES'
      IMENU(6) ='-----------------------'
      IMENU(7) ='PLOTTING...'
      IMENU(8) ='- PROFILE (DETAILED)'
      IMENU(9) ='- PROFILE (QUICK)'
      IMENU(10)='- ROUGHNESS & RUT-DEPTH'
      IMENU(11)='- RAW DATA'
      IMENU(12)='-----------------------'
      IMENU(13)='PRINT NUMERICS'
      IMENU(14)='-----------------------'
      IMENU(15)='BACK TO PREVIOUS MENU...'

C     SET DEFAULT TO 1ST ITEM(SELECT FILE)
      IDEF=1
      FINIT=0
      DR = 'D'
      IF (TINIT .EQ. 0) DR = 'C'
```

```
C     GET SELECTION
50    CONTINUE
      CALL MENU(IMENU,MI+1,MA,IDEF,IRET)
      IDEF = IRET
      GOTO (100,110,200,300,900,900,400,410,500,600,900,800,900,
     &             850) IRET

*  Open test or bounce file (100, 110)

100   PFILE = ' :*.DTA'
      GO TO 120
110   PFILE = ' :*.BNC'
120   CALL DRVSEL(DR)
      IDR = ICHAR (DR) - 64
      IF (IDR .GE. 4 .AND. TINIT .EQ. 0) CALL LOADT
      PF(1)=DR
      CALL FSEL(PFILE,FINIT,JUNK)
      IF(FINIT .EQ. 0)GOTO 50

      IF (OPENFL) THEN
        CALL UPDSET (HANDLE)
        CALL HCLOSE (HANDLE, IER)
      END IF

*   Read setup from file and verify choice. Access is set for 2 = R/W.

      CALL ADDNUL(PFILE,16)
      ACCESS=2
      CALL HOPEN(PFILE,HANDLE,ACCESS,IER)
      CALL SUBNUL(PFILE,16)
      CALL HREAD(HANDLE,SET,2048,RBYTES,IER)
      CALL TSTDIS

      IF (TSTTYP .EQ. 7) THEN
        CALL WAITKY
        I = 0
      ELSE
        CALL SETCUR(23,0)
        WRITE(*,'(''IS THIS THE FILE YOU WANTED? ''\)')
        I=1
        CALL YESNO(I,23,29,IRET)
      END IF

      IF (I .EQ. 0) THEN
        CALL HCLOSE(HANDLE,IER)
        OPENFL = .FALSE.
      ELSE
        OPENFL = .TRUE.
      END IF

*  Enable menu options based on TSTTYP and OPENFL.

  125 CONTINUE
      DO 130 I = 5, 12
  130 MA(I) = 0
      MA(3) = 0
      IF (.NOT. OPENFL) GO TO 50
      IF (TSTTYP .EQ. 4) THEN
```

```
        MA(10) = 1
      ELSE
        DO 140 I = 7, 9
  140     MA(I) = 1
        MA(12) = 1
      END IF

      IF (TSTTYP .EQ. 0 .OR. TSTTYP .EQ. 3 .OR. TSTTYP .EQ. 1
     &                .OR. TSTTYP .EQ. 5) THEN
        MA(3) = 1
        MA (10) = 1
      ELSE IF (TSTTYP .EQ. 4) THEN
        MA(3) = 1
      END IF
      GOTO 50

*  Check raw data for saturation.

200   CALL CLRSCR
      CALL CHKSAT (HANDLE, 0)
      GO TO 125

*  Option to compute profile for a list of files.

  300 IF (OPENFL) THEN
        CALL UPDSET (HANDLE)
        CALL HCLOSE(HANDLE,IER)
        OPENFL = .FALSE.
      END IF
      CALL BATCH (DR)
      GO TO 125

*  Plot elevation profile.

400   QNDPLT = .FALSE.
      GO TO 420
410   QNDPLT = .TRUE.
420   IF (TSTTYP .NE. 2 .AND. TSTTYP .NE. 6)
     &                 CALL GOAHED (HANDLE)
      IF (TSTTYP .EQ. 2 .OR. TSTTYP .EQ. 6)
     &             CALL PLTELV (HANDLE, QNDPLT)
      GO TO 125

*  Plot rut-depth, roughness, speed

500   IF (TSTTYP .NE. 2 .AND. TSTTYP .NE. 6)
     &                     CALL GOAHED (HANDLE)
      IF (TSTTYP .EQ. 2 .OR. TSTTYP .EQ. 6) CALL PLTRUT (HANDLE)
      GO TO 125

C     PLOT RAW DATA
600   CALL PLTRAW (HANDLE)
      GOTO 50

*  Print numerics

800   IF (TSTTYP .NE. 2 .AND. TSTTYP .NE. 6)
     &                     CALL GOAHED (HANDLE)
```

```
      IF (TSTTYP .EQ. 2 .OR. TSTTYP .EQ. 6) CALL PRTNUM (HANDLE)
      GO TO 125

C     RETURN TO MAIN PROGRAM
850   IF (OPENFL) THEN
        CALL UPDSET (HANDLE)
        CALL HCLOSE(HANDLE,IER)
      END IF
      CALL RDSET
      RETURN

900   CONTINUE
      END
$PAGE
```

```
      SUBROUTINE GOAHED(HANDLE)
*********************************************************************
*  This warns the user that some processing needs to be done and that
*  it might take a few minutes.  If the user answers yes, it processes
*  the data.
$INCLUDE:'SETCOM'
      INTEGER*2 HANDLE

      CALL BEEP
      CALL SETCUR (20,0)
      WRITE (*,9000)
9000  FORMAT('THIS FILE HAS RAW DATA AND NEEDS TO BE PROCESSED,'\)
      CALL SETCUR (21,0)
      WRITE (*,9010)
9010  FORMAT('WHICH MIGHT TAKE A MINUTE OR TWO.  IS THIS OK?'\)
      I = 1
      CALL YESNO (I, 21, 48, IRET)
      IF (I .EQ. 0) RETURN
      CALL CLRSCR
      IF (TSTTYP .LE. 1) THEN
        CALL CHKSAT (HANDLE, 3)
        DO 10 I = 18,23
10      CALL CLRLIN (I)
        IF (.NOT. ITSOK) RETURN
      END IF
      IF (TSTTYP .EQ. 3 .OR. TSTTYP .EQ. 5) CALL PRFCMP (HANDLE)
      RETURN
      END
```

PROFMAIN.FOR

```
$TITLE:'PROFILE MAIN'
$STORAGE:2
$NOFLOATCALLS
$INCLUDE:'BUFCOM'
$INCLUDE:'SETCOM'
$INCLUDE:'STATCOM'

      CHARACTER*32 MMENU(15)
      INTEGER*2 MAVAIL(15)
5     CALL LOGO
      CALL SETCUR(0,0)
      WRITE(*,'(A\)')' '
C     INITIALIZE TIMER BOARD AND ANALOG CONTROL

      CALL INITIO

C     INITIALIZE SETUP ARRAY AND STATUS CONTROL BLOCK
      CALL INITP

C     RESTORE ANALOG TO LAST CONDITION
      CALL RESTOR

C     CHECK A/D
      CALL CLRSCR
      CALL SETCUR(12,0)
      WRITE(*,'(A,A\)')'(DO NOT ANSWER YES UNLESS THE INSTRUMENTS ',
     &      'ARE CONNECTED)'

      CALL SETCUR(10,0)
      I=0
      WRITE(*,'(A\)')'PERFORM A/D CHECK?'
      CALL YESNO(I,10,20,IRET)
      IF(I .EQ. 1)CALL ADCHECK

      DO 10 I=1,11
10    MAVAIL(I)=1
      MITEMS=12
      MAVAIL(9)=0
      MAVAIL(3)=0
      MAVAIL(6)=0

      MMENU(1)= 'PROFILOMETER MAIN MENU'
      MMENU(2)= 'MAKE ROAD MEASUREMENTS'
      MMENU(3)= 'VIEW AND PROCESS DATA'
      MMENU(4)= '----------------------------'
      MMENU(5)= 'LOAD NEW DATA TAPE'
      MMENU(6)= 'UNLOAD DATA TAPE'
      MMENU(7)= '----------------------------'
      MMENU(8)= 'CONFIGURE TRANSDUCERS'
      MMENU(9)= 'EXERCISE INPUT/OUTPUT SYSTEM'
      MMENU(10)='----------------------------'
      MMENU(11)='DISPLAY LOGO'
      MMENU(12)='QUIT'
```

```
100     CALL MENU(MMENU,MITEMS,MAVAIL,1,IRET)
        CALL CLRSCR
        CALL SETCUR(0,0)
        GOTO (500,1000,100,1500,2000,100,2500,3000,100,4000,4500)IRET

C       MAKE ROAD MEASUREMENTS

500     CALL MEASURE
        GOTO 100

C       PROCESS DATA

1000    CALL PROCESS
        GOTO 100

C       LOAD DATA TAPE
1500    CALL LOADT
        GOTO 100

C       UNLOAD DATA TAPE
2000    CALL UNLDT
        GOTO 100

C       SETUP TRANSDUCERS
2500    CALL SETUPS
        GOTO 100

C       EXERCISE INPUT/OUTPU SYSTEM
3000    CALL IOEX
        GOTO 100

C       DISPLAY LOGO.
4000    CALL LOGO
        GO TO 100

4500    CONTINUE
        IF(TINIT .EQ. 1)THEN
        CALL SETCUR(10,0)
        WRITE(*,'(A\)')'YOU FORGOT TO UNLOAD THE TAPE'
        CALL SETCUR(11,0)
        WRITE(*,'(A\)')'DO NOT TURN POWER OFF UNTIL TAPE IS REMOVED'
        CALL UNLDT
        ENDIF
        STOP 'RESTART THIS PROGRAM BY TYPING "PROFILE"'
        END
```

PRTNUM.FOR

```
$TITLE:'SUBROUTINE PRTNUM'
$NOFLOATCALLS
$STORAGE:2

      SUBROUTINE PRTNUM (HANDLE)
*********************************************************************
*
*
$INCLUDE:'SETCOM'
$INCLUDE:'BUFCOM'
$INCLUDE:'STATCOM'
      CHARACTER*1 DR, DR2
      CHARACTER*8 FN
      CHARACTER*3 EXT
      CHARACTER*9 N(6),U(6),DASH
      CHARACTER*16 PRTFNM
      CHARACTER*25 DASH1

      INTEGER*2 HANDLE
      INTEGER*4 OFFSET, NSMP, II, JJ, ICH
      LOGICAL PRTLOG, LFL, LSCR, LLPT, IEXIST
      REAL PRTVAR(6)
      COMMON/PRSTAT/ PRTLOG(6), LFL, LSCR, LLPT, PRT1ST

*  If this is the first time (PRT1ST <> -1.) then set defaults

      IF (PRT1ST .NE. -1.) THEN
        DASH = '---------'
        DASH1=   '-----------------------------'
        PRT1ST = -1.
        EXT = '   '
        DR = 'C'
        LFL = .FALSE.
        LSCR = .TRUE.
        LLPT = .TRUE.

        DO 10 I = 1, 6
10      PRTLOG(I) = .FALSE.

        IF (LPROF) PRTLOG(1) = .TRUE.
        IF (RPROF) PRTLOG(2) = .TRUE.
        PRTLOG(3) =  .FALSE.
        IF (LRUT) PRTLOG(4) =  .TRUE.
        IF (CRUT) PRTLOG(5) =  .TRUE.
        IF (RRUT) PRTLOG(6) = .TRUE.

        N(1) = 'L. IRI'
        N(2) = 'R. IRI'
        N(3) = 'SPEED'
        N(4) = 'L. RUT'
        N(5) = 'C. RUT'
        N(6) = 'R. RUT'

        U(1) = UNITS (11)
```

```
        U(2) = UNITS(11)
        U(3) = UNITS(3)
        U(4) = UNITS(1)
        U(5) = UNITS(1)
        U(6) = UNITS(5)
      END IF

*   Prepare screen.  Start with channel names and status.

      CALL WRTSCR ('PRTSCR.     ')

      WRITE (*,'(A\)') ' '
      IF (LPROF) THEN
        CALL SETCUR (4,0)
        WRITE (*,'(A\)') N(1)
        CALL PUTYN (PRTLOG(1),4,18)
      END IF

      IF (RPROF) THEN
        CALL SETCUR (5,0)
        WRITE (*,'(A\)') N(2)
        CALL PUTYN (PRTLOG(2),5,18)
      END IF

      IF (LPROF .OR. RPROF) THEN
        CALL SETCUR (6,0)
        WRITE (*,'(A\)') N(3)
        CALL PUTYN (PRTLOG(3),6,18)
      END IF

      IF (LRUT) THEN
        CALL SETCUR (4,40)
        WRITE (*,'(A\)') N(4)
        CALL PUTYN (PRTLOG(4),4,53)
      END IF

      IF (CRUT) THEN
        CALL SETCUR (5,40)
        WRITE (*,'(A\)') N(5)
        CALL PUTYN (PRTLOG(5),5,53)
      END IF

      IF (RRUT) THEN
        CALL SETCUR (6,40)
        WRITE (*,'(A\)') N(6)
        CALL PUTYN (PRTLOG(6),6,53)
      END IF

*   Status of device switches and file name.

      CALL PUTYN (LSCR,8,22)
      CALL PUTYN (LLPT,8,38)
      CALL PUTYN (LFL,8,56)
      CALL FNMAKE (DR2,FN,EXT,PFILE,1)
      EXT = 'NUM'
      CALL SETCUR (9,56)
      WRITE (*,'(A,A,A,A,A\)') DR,':',FN,'. ',EXT
```

```
*  Print start, stop, increment.

      CALL SETCUR (12,18)
      WRITE (*,'(A\)') UNITS(10)
      CALL SETCUR (12,42)
      WRITE (*,'(A\)') UNITS(10)
      CALL SETCUR (12,71)
      WRITE (*,'(A\)') UNITS(10)

      XLL = 0.
      XUL = NSRTOT * DXTRIM
      IF (PSTART .LT. XLL .OR. PSTART .GT. XUL) PSTART = XLL
      IF (PSTOP .LT. XLL .OR. PSTOP .GT. XUL) PSTOP = XUL
      IF (PINC .LE. DXTRIM .OR. PINC .GT. XUL) PINC = XUL / 10.
      CALL SETCUR (12,7)
      WRITE (*,'(F10.2\)') PSTART
      CALL SETCUR (12,31)
      WRITE (*,'(F10.2\)') PSTOP
      CALL SETCUR (12,60)
      WRITE (*,'(F10.2\)') PINC

*  There are 14 edit field on the screen, each with a line number.
*********************************************************************

100   IF (.NOT. LPROF) GO TO 200
      CALL YESNOL (PRTLOG(1),4,18,IRET)
      GOTO (200,100,200,100,200,100,400,100,100,2000) IRET + 1

200   IF (.NOT. RPROF) GO TO 300
210   IF (.NOT. RPROF) GO TO 100
      CALL YESNOL (PRTLOG(2),5,18,IRET)
      GOTO (300,100,300,100,300,200,500,200,200,2000) IRET + 1

300   IF ((.NOT. RPROF) .AND. (.NOT. LPROF)) GO TO 700
      CALL YESNOL (PRTLOG(3),6,18,IRET)
      GOTO (700,210,700,210,700,300,610,300,300,2000) IRET + 1

400   IF (.NOT. LRUT) GO TO 500
      CALL YESNOL (PRTLOG(4),4,53,IRET)
      GOTO (500,400,500,400,500,100,400,400,400,2000) IRET + 1

500   IF (.NOT. CRUT) GO TO 600
510   IF (.NOT. CRUT) GO TO 400
      CALL YESNOL (PRTLOG(5),5,53,IRET)
      GOTO (600,400,600,400,600,200,500,510,510,2000) IRET + 1

600   IF (.NOT. RRUT) GO TO 900
610   IF (.NOT. RRUT) GO TO 510
      CALL YESNOL (PRTLOG(6),6,53,IRET)
      GOTO (900,510,900,510,900,300,600,610,610,2000) IRET + 1

700   CALL YESNOL (LSCR,8,22,IRET)
      GOTO (1200,300,1200,300,1200,700,800,700,700,2000) IRET + 1

800   CALL YESNOL (LLPT,8,38,IRET)
      GOTO (1300,300,1300,300,1300,700,900,900,800,2000) IRET + 1

900   CALL YESNOL (LFL,8,56,IRET)
```

```
      GOTO (1000,610,1000,610,1000,800,900,900,900,2000) IRET + 1

1000  IF (.NOT. LFL) GO TO 1400
1010  IF (.NOT. LFL) GO TO 900
      CALL GETSTR (DR,1,9,56,IRET)
      I=0
      DO 1020 J=1, 7
      IF (DR .EQ. CHAR(J + 64)) I = 1
1020  CONTINUE
      IF (I .EQ. 0) THEN
        CALL BEEP
        DR = 'C'
        CALL SETCUR (9,56)
        WRITE (*,'(A\)') DR
        GO TO 1010
      ELSE
        CALL GFILE (DR,FN,EXT,IEXIST,9,58,IRET)
        IF (IEXIST) THEN
          CALL SETCUR (10,20)
          WRITE (*,9010)
9010    FORMAT('<That file already exists and will be overwritten.>')
        END IF
      END IF
      GOTO (1400,900,1400,900,1400,800,1100,1010,1010,2000) IRET + 1

1100  IF (.NOT. LFL) GO TO 1400
1110  IF (.NOT. LFL) GO TO 900
      CALL GFILE (DR,FN,EXT,IEXIST,9,58,IRET)
      IF (IEXIST) THEN
        CALL SETCUR (10,20)
        WRITE (*,9010)
      ELSE
        CALL CLRLIN (10)
      END IF
      GOTO (1400,900,1400,900,1400,1010,1110,1110,1110,2000) IRET + 1

1200  CALL GETR (PSTART,XLL,XUL,12,7,10,'(F10.2\)',IRET)
      GOTO (1300,700,1200,700,1200,1200,1300,1200,1200,2000) IRET + 1

1300  CALL GETR (PSTOP,XLL,XUL,12,31,10,'(F10.2\)',IRET)
      GOTO (1400,800,1300,800,1300,1200,1400,1300,1300,2000) IRET + 1

1400  CALL GETR (PINC,XLL,XUL,12,60,10,'(F10.2\)',IRET)
      GOTO (1400,1110,1400,1110,1400,1300,1400,1400,1400,2000) IRET+1

************************************************************************
*

*  Quit the subroutine if there's nothing to print.

2000  CONTINUE
      IF ((.NOT. LFL) .AND. (.NOT. LSCR) .AND. (.NOT. LLPT)) RETURN
      IF (PSTART .GE. PSTOP) RETURN
      IF (PINC .LE. 0) RETURN
      IF ((.NOT. PRTLOG(1)) .AND. (.NOT. PRTLOG(2)) .AND.
     &      (.NOT. PRTLOG(3)) .AND. (.NOT. PRTLOG(4)) .AND.
     &      (.NOT. PRTLOG(5)) .AND. (.NOT. PRTLOG(6))) RETURN
```

```
*   Set names of units from SETCOM data

      U(1) = UNITS(11)
      U(2) = UNITS(11)
      U(3) = UNITS(3)
      U(4) = UNITS(1)
      U(5) = UNITS(1)
      U(6) = UNITS(5)

*   Open printer and/or disk file

      IF (LLPT) THEN
        OPEN (6,FILE='LPT1')
        WRITE (6, *) CHAR (12)
      END IF

      IF (LFL) THEN
        CALL FNMAKE (DR,FN,EXT,PRTFNM,0)
        IF (IEXIST) THEN
          OPEN (7,FILE=PRTFNM,STATUS='OLD')
        ELSE
          OPEN (7,FILE=PRTFNM,STATUS='NEW')
        END IF
      END IF

*   Clear screen

      IF (LSCR) THEN
        CALL CLRSCR
        CALL SETCUR (0,0)
      END IF

*   Everything's in 3's...

      IF (LSCR) WRITE (*,'(4X,A,A,A,7X\)') CHID(10),' - ',UNITS(10)
      IF (LLPT) WRITE (6,'(4X,A,A,A,7X\)') CHID(10),' - ',UNITS(10)
      IF (LFL) WRITE (7,'(4X,A,A,A,7X\)') CHID(10),' - ',UNITS(10)

      DO 2020 I = 1, 6
        IF (LSCR .AND. PRTLOG(I)) WRITE (*,'(A\)') N(I)
        IF (LLPT .AND. PRTLOG(I)) WRITE (6,'(A\)') N(I)
        IF (LFL .AND. PRTLOG(I)) WRITE (7,'(A\)') N(I)
2020  CONTINUE
      CALL PRTLF (LSCR,LLPT,LFL)

      IF (LSCR) WRITE(*,9050)
9050  FORMAT(    '         FROM...         TO          '\)
      IF (LLPT) WRITE(6,9050)
      IF (LFL) WRITE(7,9050)

      DO 2030 I = 1, 6
        IF (LSCR .AND. PRTLOG(I)) WRITE (*,'(A\)') U(I)
        IF (LLPT .AND. PRTLOG(I)) WRITE (6,'(A\)') U(I)
        IF (LFL .AND. PRTLOG(I)) WRITE (7,'(A\)') U(I)
2030  CONTINUE
      CALL PRTLF (LSCR,LLPT,LFL)

      IF (LSCR) WRITE (*,'(A\)') DASH1
```

```
      IF (LLPT) WRITE (6,'(A,A\)') ' ',DASH1
      IF (LFL) WRITE (7,'(A\)') DASH1

      DO 2040 I = 1, 6
        IF (LSCR .AND. PRTLOG(I)) WRITE (*,'(A\)') DASH
        IF (LLPT .AND. PRTLOG(I)) WRITE (6,'(A\)') DASH
        IF (LFL .AND. PRTLOG(I)) WRITE (7,'(A\)') DASH
2040  CONTINUE
      CALL PRTLF (LSCR,LLPT,LFL)

*  Loop here until all numerics are printed

      IF (PINC .LT. DXTRIM) PINC = DXTRIM
      X2 = PSTART
2050  X1 = X2
      X2 = X1 + PINC
      IF (X2 .GT. PSTOP) X2 = PSTOP

*  Read some data from the file

      OFFSET = X1/DXTRIM
      NSMP = X2/DXTRIM
      NSMP = NSMP - OFFSET + 1
      IF (NSMP .LE. 1) NSMP = 2
      IF (OFFSET + NSMP .GT. NSRTOT) NSMP = NSRTOT - OFFSET
      IF (NSMP .LE. 1) GO TO 2080
      CALL RDTAPD (HANDLE, PCBUFR, 2, OFFSET, NSMP, IERR)

*  Average the variables in the file.

      DO 2060 ICH = 1,NCHRUT
        I = ICH
        IF (I .EQ. ILIRI .OR. I .EQ. IRIRI) THEN
          IF (I .EQ. ILIRI) I2 = 1
          IF (I .EQ. IRIRI) I2 = 2
          PRTVAR (I2) = SCLFRI * (PCBUFR (ICH + (NSMP - 1) * NCHRUT)
     &        - PCBUFR (ICH)) / (DXTRIM * NSMP)
        ELSE
          SUM = PCBUFR (ICH)
          II = ICH
          DO 2055 JJ = 2, NSMP
            II = II + NCHRUT
            SUM = PCBUFR (II) + SUM
2055      CONTINUE

          IF (I .EQ. IVEL) THEN
            I2 = 3
          ELSE IF (I .EQ. ILR) THEN
            I2 = 4
          ELSE IF (I .EQ. ICR) THEN
            I2 = 5
          ELSE IF (I .EQ. IRR) THEN
            I2 = 6
          END IF
          PRTVAR (I2) = SUM / NSMP
        END IF
2060  CONTINUE
```

```
*  Print them

      IF (LSCR) WRITE (*,9000) X1,X2
      IF (LLPT) WRITE (6,9000) X1,X2
      IF (LFL) WRITE (7,9000) X1,X2
9000  FORMAT (F10.2,' - ',F10.2,1X\)

      DO 2070 I = 1, 6
        IF (LSCR .AND. PRTLOG(I)) WRITE (*,'(F9.2\)')PRTVAR(I)
        IF (LLPT .AND. PRTLOG(I)) WRITE (6,'(F9.2\)')PRTVAR(I)
        IF (LFL .AND. PRTLOG(I)) WRITE (7,'(F9.2\)')PRTVAR(I)
2070  CONTINUE
      CALL PRTLF (LSCR,LLPT,LFL)
      IF (X2 .LT. PSTOP) GO TO 2050

2080  CONTINUE

      IF (LLPT) WRITE (6,*) CHAR (12)
      CLOSE (6)
      CLOSE(7)

      WRITE (*,'(A\)') ' '
      IF (LSCR) CALL WAITKY
      RETURN
      END

$PAGE
```

```
      SUBROUTINE PRTLF (LSCR,LLPT,LFL)
****************************************************************
*  Add carriage returns after each line

      LOGICAL LSCR,LLPT,LFL
      IF (LSCR) WRITE (*,*) ' '
      IF (LLPT) WRITE (6,*) ' '
      IF (LFL) WRITE (7,*) ' '
      RETURN
      END
$PAGE
```

```
      SUBROUTINE PUTYN (L, IROW, ICOL)
************************************************************
*  Put 'Y' or 'N' in position IROW, ICOL, based on L

      LOGICAL L
      CALL SETCUR (IROW, ICOL)
      IF (.NOT. L) WRITE (*,'(A\)') 'N'
      IF (L) WRITE (*,'(A\)') 'Y'
      RETURN
      END

$PAGE
```

```
      SUBROUTINE YESNOL (IL,IR,ICOL,IRET)
*****************************************************************
*  Get Yes/No anser and set logical variable.

      LOGICAL IL
      I=0
      IF (IL) I = 1
      CALL YESNO (I,IR,ICOL,IRET)
      IL = .FALSE.
      IF (I .EQ. 1) IL = .TRUE.
      RETURN
      END
```

PULSE.FOR

```
$STORAGE:2
$NOFLOATCALLS

      SUBROUTINE PULSE

C     THIS SUBROUTINE CHECKS THE ELAPSED DISTANCE VERSUS
C     A KNOWN DISTANCE TRAVELLED
C
$INCLUDE:'BUFCOM'
$INCLUDE:'SETCOM'
$INCLUDE:'IOPARMS'
      INTEGER*4 MAXP,PASS,JJ,KK,II
      INTEGER*2 DONE,CONV,AD(5),HIGH,QA(2)
      EQUIVALENCE (JJ,QA)

      CALL CLRSCR
      CALL SETCUR(2,0)
      WRITE(*,'(A\)')'THIS IS A TEST TO COMPARE A MEASURED '
      CALL SETCUR(3,0)
      WRITE(*,'(A\)')'DISTANCE WITH THE DISTANCE PROVIDED BY THE'
      CALL SETCUR(4,0)
      WRITE(*,'(A\)')'WHEEL PULSER'
      CALL SETCUR(5,0)
      WRITE(*,'(A\)')'DO YOU WISH TO CONTINUE ?'
      I=1
      CALL YESNO(I,5,26,IRET)
      IF(I .EQ. 0)RETURN
      CALL SETCUR(8,0)
      WRITE(*,'(A\)')'ENTER THE DISTANCE YOU WILL TRAVEL BETWEEN'
      CALL SETCUR(9,0)
      WRITE(*,'(A\)')'KEY PRESSES =            FEET'
      TRUED=5280.
      CALL GETR(TRUED,500.,26400.,9,14,8,'(F8.1\)',IRET)
C     SET CONVERSION PARAMETERS
      DONE=0
      PASS=0
      CONV=3
      MAXP=131072/CONV
      CALL PHYSAD(IBUF(1),JJ)
      AD(1)=1
      AD(2)=2
      AD(3)=4
      AD(4)=CONV
      AD(5)=0
C     SET DT CLOCK
      F=20000.
      CALL DTCLOCK(F)
      CALL SETAD(AD)
C
C     CALCULATE COUNTER VALUE
C
      D=12.0/4.
      IDIV=NINT(D/XDUCGN(9))
      ITMODE=#0221
```

```
      F=100.
      T=IDIV*XDUCGN(9)/12.0
C
C     SET COUNTER #4 FOR APPROPRIATE MODE
300   CALL IOUTB(#C8,TIMERC)
      CALL IOUTB(4,TIMERC)
      M=8
      L=#00FF
      LOW=IAND(ITMODE,L)
      HIGH=ISHFTR(ITMODE,M)
      CALL IOUTB(LOW,TIMERD)
      CALL IOUTB(HIGH,TIMERD)
      LOW=IAND(IDIV,L)
      HIGH=ISHFTR(IDIV,M)
      CALL IOUTB(#0C,TIMERC)
      CALL IOUTB(LOW,TIMERD)
      CALL IOUTB(HIGH,TIMERD)
C
C     START COUNTER 4
      CALL IOUTB(#68,TIMERC)
C     SET FILTER CLOCK
      CALL FILCLK(F)
C     START
      CALL PULTST(PASS,DONE,JJ,CONV,MAXP)
C
C     WAIT FOR KEY
C
      CALL SETCUR(12,0)
      WRITE(*,'(A\)')' HIT ANY KEY TO START'
      CALL KCLEAR
350   I=IGKEY()
      IF(I .EQ. 0)GOTO 350
      CALL GTIME(I1,I2,I3)
      TIM1=I1*3600.+I2*60.+I3
      CALL SETCUR(12,16)
      WRITE(*,'(A\)')'STOP '
      CALL SETCUR(14,0)
      WRITE(*,'(A\)')' ELAPSED DISTANCE=          FEET'
      CALL SETCUR(15,0)
      WRITE(*,'(A\)')' VELOCITY=      MPH'
C
C     ENABLE BOARD INTS+FLIP FLOP
      CALL IOUTB(1,CNTRL)
      CALL IOUTB(0,INTE)
      CALL KCLEAR
500   IF(DONE .NE. 0) GOTO 600
      TA=T*PASS
      IF (PASS .LT. 2)THEN
      V=0.0
      ELSE
      V=FLOAT(IBUF((PASS-2)*3+1))*GAIN(3)-ZDATA(3)
      ENDIF
      CALL SETCUR(14,18)
      WRITE(*,9520)TA
9520  FORMAT(,F8.1\)
      CALL SETCUR(15,11)
      WRITE(*,'(F4.1\)')V
      I=IGKEY()
```

```
      IF( I .EQ. 0) GOTO 500
      CALL GTIME(I1,I2,I3)
600   TIM2=I1*3600.+I2*60.+I3
      IF(DONE .EQ. 0)DONE=-1
      I=INPB(#21)
      I=IOR(I,4)
      CALL IOUTB(I,#21)
      CALL IOUTB(0,INTD)
      IF(DONE .GT. 0) GOTO 700
      TA=PASS*T
C     CALCULATE AVERAGE VELOCITY
      IF(PASS .EQ. 0)RETURN
      VT=0.0
      TSUM=0.0
      DO 1000 II=1,PASS
      V=FLOAT(IBUF((II-1)*3+1))*GAIN(3)-ZDATA(3)
      IF(ABS(V) .LT. 0.001)RETURN
      DT=T/V
      TSUM=TSUM+DT
      VT=VT+V*DT
1000  CONTINUE
      AVVEL=VT/TSUM
      DTIME=TIM2-TIM1
      AVVELT=TRUED/DTIME*60./88.
      CALL CLRLIN(14)
      CALL CLRLIN(15)
      CALL SETCUR(15,0)
      WRITE(*,'(A\)')'PULSER DISTANCE     TRUE DISTANCE'
      CALL SETCUR(16,0)
      WRITE(*,'(F8.1,13X,F8.1\)')TA,TRUED
      CALL SETCUR(18,0)
      WRITE(*,'(A\)')'MEASURED VELOCITY    TRUE VELOCITY'
      CALL SETCUR(19,0)
      WRITE(*,'(4X,F8.2,9X,F8.2\)')AVVEL,AVVELT
      CALL WAITKY
      RETURN
700   WRITE(*,*)' A/D ERROR'
      WRITE(*,*)' PASSES=',PASS
      CALL WAITKY
      RETURN
      END
```

PULSETST.ASM

```
      TITLE PULSE TEST
      PAGE  ,132
DATA  SEGMENT      PUBLIC      'DATA'
PASSA DW     ?                 ;PASS ADDRESS
      DW     ?
DONEA DW     ?                 ;DONE ADDRESS
      DW     ?
PHSAD DW     ?                 ;PHYSICAL BUFFER ADDRESS
PHSADH       DW     ?
CONV  DW     ?                 ;NUMBER OF CHANNNELS
MAXP  DW     ?                 ;MAX NUMBER OF PASSES
MAXPH DW     ?
DATA  ENDS
DGROUP       GROUP DATA
CODE  SEGMENT      'CODE'
      ASSUME       CS:CODE,DS:DGROUP,SS:DGROUP;
DSSAVE       DW     ?                ;SAVE DS
;
;     PULTST(PASS,DONE,PHSAD,CONV,MAXP)
;            PASS=INT*4 PASS #
;            DONE=INT*2 DONE FLAG -1=DONE 1=ERROR 0=NOT DONE
;            PHSAD=INT*4 PHYSICAL BUFFER ADDRESS
;            CONV=# OF CONVERSIONS
;            MAXP=MAX NUMBER OF PASSES
;
PUBLIC       PULTST
PULTST       PROC  FAR
      PUSH  BP
      MOV   BP,SP
      MOV   DSSAVE,DS          ;SAVE DS FOR INTERRUPT
      LES   BX,DWORD PTR [BP+6]      ;GET MAXP ADDRESS
      MOV   AX,ES:[BX]         ;GET LOW WORD
      MOV   MAXP,AX                  ;SAVE IT
      MOV   AX,ES:[BX]+2             ;GET HIGH WORD
      MOV   MAXPH,AX           ;SAVE IT
      LES   BX,DWORD PTR [BP+10]     ;GET CONV ADDRESS
      MOV   AX,ES:[BX]         ;GET # OF CHANNELS
      MOV   CONV,AX                  ;SAVE IT
      LES   BX,DWORD PTR [BP+14]     ;GET PHYSAD ADDRESS
      MOV   AX,ES:[BX]         ;GET LOW WORD
      MOV   PHSAD,AX           ;SAVE IT
      MOV   AX,ES:[BX]+2             ;GET HIGH WORD
      MOV   PHSADH,AX          ;SAVE IT
      LES   BX,DWORD PTR [BP+18]     ;GET DONE ADDRESS
      MOV   DONEA,BX           ;SAVE OFFSET
      MOV   DONEA+2,ES         ;SAVE SEGMENT
      LES   BX,DWORD PTR [BP+22]     ;GET PASS ADDRESS
      MOV   PASSA,BX           ;SAVE OFFSET
      MOV   PASSA+2,ES         ;SAVE SEGMENT
;
;     SET UP INTERRUPT VECTOR
;
      CLI                      ;DISABLE INTS
      PUSH  DS                 ;SAVE DS
```

```
        MOV     DX,OFFSET ISRP          ;GET VECTOR OFFSET
        PUSH    CS
        POP     DS                      ;DS=SEGMENT FOR INT ROUTINE
        MOV     AL,0AH                  ;INTERRUPT VECTOR #
        MOV     AH,25H                  ;SET VECTOR FUNCTION
        INT     21H                     ;SET IT
        POP     DS                      ;RECOVER DS
;
;       ENABLE IRQ2 ON 8259
;
        IN      AL,21H                  ;GET CURRENT MASK
        AND     AL,11111011B            ;RESET IRQ2
        OUT     21H,AL
        MOV     SP,BP
        POP     BP
        STI                             ;ENABLE INTS
        RET     20                      ;RETURN
PULTST          ENDP
;
;       EQUATES FOR ISRP
;
PCTRL EQU       307H                    ;8255 CONTROL REG
DTCOM EQU       2EDH                    ;A/D COMMAND REG
DTSTAT          EQU     2EDH            ;A/D STATUS REG
DTDATA          EQU     2ECH            ;A/D DATA REG
CWAIT EQU       4                       ;COMMAND WAIT
RWAIT EQU       5                       ;READ WAIT
CDMA  EQU       1EH                     ;A/D DMA COMMAND
CRAD  EQU       0EH                     ;A/D NON-DMA COMMAND
INTD  EQU       310H                    ;INT DISABLE ADDRESS
;
;       INTERRUPT ROUTINE FOR PULSE TEST
;
ISRP  PROC      NEAR
      CLI                               ;NO INTS
      PUSH      AX                      ;SAVE REGISTERS
      PUSH      BX
      PUSH      CX
      PUSH      DX
      PUSH      DS
      PUSH      ES
      MOV       AX,DSSAVE               ;GET DS
      MOV       DS,AX                   ;SET IT
      MOV       DX,DTSTAT               ;GET STATUS ADDRESS
      IN        AL,DX                   ;GET STATUS
      TEST      AL,80H                  ;ERROR ?
      JE        ISRPA
      JMP       DTERR                   ;YES EXIT
ISRPA:          TEST    AL,4            ;COMMAND COMPLETE?
      JNE       ISRPB
      JMP       DTERR                   ;NO=ERROR
;
;     TEST FOR PAGE OVER RUN
;
ISRPB:          MOV     CX,CONV         ;GET # OF CONV
      SHL       CX,1                    ;*2=BYTES
      MOV       BX,CX                   ;SAVE FOR DMA
      CLC
```

```
        ADD     CX,PHSAD                ;ADD TO BASE
        JC      NODMA                   ;PAGE CROSSING=NO DMA
;
;       SET DMA
;
        DEC     BX                      ;BX=2*CONV-1
        MOV     AL,45H
        OUT     11,AL                   ;SET DMA MODE
        MOV     AL,0
        OUT     12,AL                   ;RESET BYTE FLIP FLOP
        MOV     AX,PHSAD                ;GET BASE ADDRESS
        OUT     2,AL                    ;SET LOW BYTE
        MOV     AL,AH                   ;AL=HIGH BYTE
        OUT     2,AL                    ;SET HIGH BYTE
        MOV     AL,BL                   ;GET CONV
        OUT     3,AL                    ;SET LOW BYTE
        MOV     AL,BH                   ;GET HIGH BYTE
        OUT     3,AL                    ;SET IT
        MOV     AX,PHSADH               ;GET PAGE
        OUT     83H,AL                        ;SET IT
        MOV     AL,1                    ;ENABLE MASK
        OUT     10,AL
        MOV     AL,CDMA                       ;GET START COMMAND
        OUT     DX,AL                   ;START
        JMP     ISRP2                   ;GO INC PASS + EXIT
;
;       NON DMA A/D
;       CALCULATE SEGMENT AND OFFSET
;
NODMA:          MOV     AX,PHSAD              ;GET BASE
        MOV     BX,AX                   ;SAVE IT
        AND     BX,000FH                ;BX=OFFSET
        MOV     CL,4
        SHR     AX,CL                   ;SHIFT OUT OFFSET
        MOV     DX,PHSADH               ;GET PAGE
        MOV     CL,12
        SHL     DX,CL                   ;PUT PAGE IN UPPER NIBBLE
        OR      AX,DX                   ;AX=SEGMENT
        MOV     ES,AX                   ;SET SEGMENT
        MOV     DX,DTCOM                ;GET A/D COMMAND ADDR
        MOV     AL,CRAD                       ;GET COMMAND
        OUT     DX,AL                   ;START
        MOV     CX,CONV                       ;GET # OF CONV
        SHL     CX,1                    ;BYTES=*2
ADLP:   MOV     DX,DTSTAT               ;GET STATUS ADDRESS
ADLP1:          IN      AL,DX                 ;GET STATUS
        TEST    AL,RWAIT                ;BYTE READY?
        JE      ADLP1                   ;NO WAIT
        MOV     DX,DTDATA
        IN      AL,DX                   ;GET DATA BYTE
        MOV     BYTE PTR ES:[BX],AL           ;SAVE IT
        INC     BX                      ;POINT TO NEXT LOCATION
        LOOP    ADLP
;
;       UPDATE BUFFER ADDRESS
;
ISRP2:          MOV     CX,CONV                     ;GET CONV
        SHL     CX,1                    ;*2
```

```
        CLC
        ADD     PHSAD,CX            ;ADD TO BASE
        JNC     ISRP1
        INC     PHSADH                      ;PAGE=PAGE+1
;
;       INCREMENT PASS COUNT AND COMPARE TO MAXP
;
ISRP1:          LES     BX,DWORD PTR PASSA      ;ES:[BX]=PASS ADDRESS
        CLC
        ADD     WORD PTR ES:[BX],1          ;PASS=PASS+1
        JNC     ISRP3
        INC     WORD PTR ES:[BX]+2          ;INCREMENT UPPER WORD
ISRP3:          MOV     AX,MAXP                 ;GET MAXP LOW
        CMP     AX,ES:[BX]          ;LOW WORDS EQUAL
        JNE     DNCHK               ;NO-CHECK FOR DONE
        MOV     AX,MAXPH            ;GET MAXP HIGH
        CMP     AX,ES:[BX]+2
        JNE     DNCHK
;
;       SET DONE FLAG
        LES     BX,DWORD PTR DONEA          ;GET DONE ADDR
        MOV     WORD PTR ES:[BX],0FFFFH ;SET DONE
DNCHK:          LES     BX,DWORD PTR DONEA
        CMP     WORD PTR ES:[BX],0          ;DONE?
        JE      ISRPOT                      ;NO
ISRP4:          MOV     DX,INTD                 ;GET INT DISABLE ADDRESS
        OUT     DX,AL               ;DISABLE INTS
ISRPOT:         MOV     AL,0
        MOV     DX,PCTRL            ;GET INT FF ADDRESS
        OUT     DX,AL               ;RESET FLIP FLOP
        INC     AL
        OUT     DX,AL               ;RE-ENABLE IT
;
;       SIGNAL END OF INT TO 8259
        MOV     AL,20H
        OUT     20H,AL
;
;       RECOVER REGS AND EXIT
        POP     ES
        POP     DS
        POP     DX
        POP     CX
        POP     BX
        POP     AX
        IRET                        ;RETURN
;
;       ERROR-SET DONE>0
;
DTERR:          MOV     AX,1
        LES     BX,DWORD PTR DONEA              ;GET DONE ADDRESS
        MOV     WORD PTR ES:[BX],AX         ;SET DONE
        JMP     ISRP4
ISRP    ENDP
CODE    ENDS
        END
```

RDTAPD.FOR

```
$TITLE:'SUBROUTINE RDTAPD'
$STORAGE:2
$NOFLOATCALLS
***********************************************************************
      SUBROUTINE RDTAPD (HANDLE, ARRAY, WHICH, OFFSET, NSMP, IERR)
***********************************************************************
*  This subroutine reads numerical data from tape.  It allows the
*  calling program to treat the data on tape as if it were contiguous,
*  instead of the interleaved format that is actually used.
*
*  -->  HANDLE int*2   handle for tape file.
*  <--  ARRAY  real*4  array in memory that holds the data read from
*                      the tape.
*  -->  WHICH  int*2   code for data type. 1=slope profile, 2=rut
*                      stuff, 3=profile elevation.
*  -->  OFFSET int*4   number of samples to skip before 1st.
*  <->  NSMP   int*4   number of samples to read.  If NSMP is too
*                      large and goes beyond the range of data
*                      existing on tape, the subroutine will reset
*                      NSMP to the number of samples actually read.
*  <--  IERR   int*2   error return code.  0=cool.
*
*  Important variables unique to this subroutine:
*    ABSOFF int*4   number of bytes preceeding the next point.
*    BUFSIZ int*4   full-words/buffer for the selected data type.
*    FRSTUN int*4   full-words preceeding 1st point in 1st buffer.
*    ICOUNT int*4   no. of full-words that have been read so far.
*    LASTUN int*4   total no. of words preceding current buffer.
*    NBYTES int*4   number of bytes to read next.
*    NFW    int*4   number of full-words to be read.
***********************************************************************
$LARGE:ARRAY
      REAL*4 ARRAY (*)
      INTEGER*2 HANDLE, WHICH, IERR
      INTEGER*4 ABSOFF, OFFSET, NSMP, LASTUN, ICOUNT, NBYTES, NFW,
     &          FRSTUN, BUFSIZ
$INCLUDE:'SETCOM'
*
*  Set local pointers and size variables.  change NSMP if it is too
*  large.
*
*     WRITE (6,*) 'STARTING IN RDTAPD. HANDLE, WHICH,OFFSET,NSMP=',
*    &      HANDLE,WHICH,OFFSET,NSMP
      IF (WHICH .EQ. 1) THEN
        BUFSIZ = NPRFFW
        FRSTUN = MOD (OFFSET * NCHPRF, NPRFFW)
        LASTUN = (OFFSET * NCHPRF / NPRFFW) * NBUFFW
        NFW = NSMP * NCHPRF
        IF (OFFSET + NSMP .GT. NSPTOT) NSMP = NSPTOT - OFFSET
      ELSE IF (WHICH .EQ. 2) THEN
        BUFSIZ = NRUTFW
        FRSTUN = MOD (OFFSET * NCHRUT, NRUTFW)
        LASTUN = (OFFSET * NCHRUT / NRUTFW) * NBUFFW + NPRFFW
        NFW = NSMP * NCHRUT
```

```
        IF (OFFSET + NSMP .GT. NSRTOT) NSMP = NSRTOT - OFFSET
      ELSE
        BUFSIZ = NELVFW
        FRSTUN = MOD (OFFSET * NCHPRF, NELVFW)
        LASTUN = (OFFSET * NCHPRF / NELVFW) * NBUFFW + NPRFFW + NRUTFW
        NFW = NSMP * NCHPRF
        IF (OFFSET + NSMP .GT. NSRTOT) NSMP = NSRTOT - OFFSET
      END IF
*
      ABSOFF = 4 * (LASTUN + FRSTUN)
*
*  Read data from first buffer for case that all data are in first
*  buffer
*
      IF (FRSTUN + NFW .LE. BUFSIZ) THEN

        NBYTES = NFW * 4

*     WRITE (6,*) 'IN RDTAPD, FOR CASE OF ONLY 1 BUFFER.'
*     WRITE (6,*) 'ABSOFF, NBYTES =',ABSOFF,NBYTES

        CALL RDTAPE (HANDLE, ARRAY, ABSOFF, NBYTES, IERR)

*     WRITE (6,*) 'ARRAY RETURNED:'
*     WRITE (6,*) (ARRAY(I),I=1,NSMP)

        RETURN
      END IF
*
*  Read data from first buffer for case that the data continue into
*  the next buffer
*
      NBYTES = (BUFSIZ - FRSTUN) * 4
      CALL RDTAPE (HANDLE, ARRAY, ABSOFF, NBYTES, IERR)
      ICOUNT = BUFSIZ - FRSTUN
*
*  Loop to read data from the rest of the buffers.  Check each time
*  for end of data.
*
   30 CONTINUE
      LASTUN = LASTUN + NBUFFW
      ABSOFF = LASTUN * 4
      IF (ICOUNT + BUFSIZ .LE. NFW) THEN
        NBYTES = NBUFFW * 4
        CALL RDTAPE (HANDLE, ARRAY (ICOUNT + 1), ABSOFF, NBYTES, IERR)
        ICOUNT = ICOUNT + BUFSIZ
        GO TO 30
      END IF
*
*  Last buffer.
*
      NBYTES = (NFW - ICOUNT) * 4
      CALL RDTAPE (HANDLE, ARRAY (ICOUNT + 1), ABSOFF, NBYTES, IERR)
      RETURN
      END
```

RDWRTAPE.FOR

```
$TITLE:'READ & WRITE TAPE ROUTINES'
$STORAGE:2
$NOFLOATCALLS
C       Subroutine for reading binary data from tape or disk file
C       -->  HANDLE  INT*2  file handle
C       <--  ARRAY   INT*2  destination array for data
C       -->  OFFSET  INT*4  Offset into file 0=start
C       -->  NBYTES  INT*4  number of bytes to read
C       <--  IER     INT*2  error return 0=no error

        SUBROUTINE RDTAPE(HANDLE,ARRAY,OFFSET,NBYTES,IER)

$INCLUDE:'HANDLES'
$LARGE: ARRAY

        INTEGER*4 NBYTES,LBYTES,IO,IP,II,NBUF,I
        INTEGER*2 ARRAY(*),IER

*      WRITE (6,*) 'JUST INTO RDTAPE.  HANDLE, OFF, NBYTES=',
*      &     HANDLE, OFFSET, NBYTES

C       ADD SETUP BYTES TO OFFSET
        IO=2048+OFFSET
        METHOD=0

C       POSITION FILE
        CALL HPOS(HANDLE,METHOD,IO,IP,IER)
        IF( IER .NE. 0)RETURN

C       CALCULATE HOW MANY 32768 BYTE BUFFERS TO READ
        BYTES=32768
        LBYTES=MOD(NBYTES,BYTES)
        NBUF=NBYTES/BYTES
        IF(LBYTES .NE. 0)THEN
          NBUF=NBUF+1
        ELSE
          LBYTES=BYTES
        ENDIF

C       READ DATA
        DO 100 I=1,NBUF
        IF(I .EQ. NBUF)BYTES=LBYTES
        II=1+(I-1)*16384
        CALL HREAD(HANDLE,ARRAY(II),BYTES,RBYTES,IER)
        IF( IER .NE. 0)RETURN
100     CONTINUE

        RETURN
        END

$PAGE
```

```
C       Subroutine for writing binary data to tape or disk file
C       -->  HANDLE  INT*2  file handle
C       -->  ARRAY   INT*2  source array for data
C       -->  OFFSET  INT*4  Offset into file 0=start
C       -->  NBYTES  INT*4  number of bytes to write
C       <--  IER     INT*2  error return 0=no error

        SUBROUTINE WRTAPE(HANDLE,ARRAY,OFFSET,NBYTES,IER)

$INCLUDE:'HANDLES'
$LARGE: ARRAY
        INTEGER*4 NBYTES,LBYTES,IO,IP,II,NBUF,I
        INTEGER*2 ARRAY(*),IER

C       ADD SETUP BYTES TO OFFSET
        IO=2048+OFFSET
        METHOD=0

C       POSITION FILE
        CALL HPOS(HANDLE,METHOD,IO,IP,IER)
        IF( IER .NE. 0)RETURN

C       CALCULATE HOW MANY 32768 BYTE BUFFERS TO WRITE
        BYTES=32768
        LBYTES=MOD(NBYTES,BYTES)
        NBUF=NBYTES/BYTES
        IF(LBYTES .NE. 0)THEN
          NBUF=NBUF+1
        ELSE
          LBYTES=BYTES
        ENDIF

C       READ DATA
        DO 100 I=1,NBUF
        IF(I .EQ. NBUF)BYTES=LBYTES
        II=1+(I-1)*16384
        CALL HWRITE(HANDLE,ARRAY(II),BYTES,RBYTES,IER)
        IF( IER .NE. 0)RETURN
100     CONTINUE

        RETURN
        END
```

RDWRTSET.FOR

```
$STORAGE:2
$NOFLOATCALLS

      SUBROUTINE RDSET

$INCLUDE:'SETCOM'
C
C     READ IN SETUP FROM DRIVE C (BUBBLE)
C
      OPEN(9,FILE='SETUP.SET',ACCESS='DIRECT',FORM='BINARY',RECL=2048)
      READ(9,REC=1)SET
      CLOSE(9)
      RETURN
      END
```

```
      SUBROUTINE WRTSET

$INCLUDE:'SETCOM'

C     WRITE SETUP ARRAY TO SETUP FILE

      OPEN(9,FILE='SETUP.SET',ACCESS='DIRECT',FORM='BINARY',RECL=2048)
      WRITE(9,REC=1)SET
      CLOSE(9)
      RETURN
      END
```

```
**********************************************************************
      SUBROUTINE UPDSET (HANDLE)
**********************************************************************
*  This updates the setup info for a file using its handle.
*
$INCLUDE:'HANDLES'
$INCLUDE:'SETCOM'
      BYTES = 2048
      OFFSET = 0
      METHOD = 0
      CALL HPOS (HANDLE, METHOD, OFFSET, POINTER, IER)
      IF (IER .NE. 0) RETURN
      CALL HWRITE (HANDLE, SET, BYTES, RBYTES, IER)
      RETURN
      END
```

RUTCMP.FOR

```
$TITLE:'SUBROUTINE RUTCMP'
$NOFLOATCALLS
$STORAGE:2
$LARGE: HL, HC, HR, RUT

      SUBROUTINE RUTCMP (HL, HC, HR, NCHRAW, NS, RUT, NCHRUT, TRIM,
     &          GAINL, GAINC, GAINR, ZL, ZC, ZR, HLLAT, HRLAT)
*********************************************************************
*  Subroutine to compute, average, and decimate a rut-depth signal.
*
*  --> HL     int*4  2-D array with left-hand height signal.
*  --> HC     int*2  2-D array with center (in rut) height signal.
*  --> HR     int*2  2-D array with right-hand height signal.
*  --> NCHRAW int*2  number of raw data channels. (HL, HC, HR are
*                    channels in the same 2-D array.)
*  --> NS     int*4  number of samples before decimation.
*  <-- RUT    real*4 2-D array for output rut-depth signal.
*  --> NCHRUT int*4  number of channels in output array.
*  --> TRIM   int*4  decimation ratio.
*  --> GAINL  real*4 gain for left-hand height signal.
*  --> GAINC  real*4 gain for center height signal.
*  --> GAINL  real*4 gain for right-hand height signal.
*  --> ZL     real*4 height of L. height sensor when it reads zero.
*  --> ZC     real*4 height of C. height sensor when it reads zero.
*  --> ZR     real*4 height of R. height sensor when it reads zero.
*  --> HLLAT  real*4 lateral distance between L. and C. sensors.
*  --> HRLAT  real*4 lateral distance between R. and C. sensors.
*********************************************************************
      INTEGER*2 HL(*), HC(*), HR(*)
      INTEGER*4 NCHRAW, NCHRUT, NS, TRIM, IL, IR, IC, IRUT, I, J
      REAL GAINL, GAINC, GAINR, ZL, ZC, ZR, RUT (*), SUM, HLLAT,
     &     HRLAT, CL, CR
*
      CL = HRLAT / (HLLAT + HRLAT)
      CR = HLLAT / (HLLAT + HRLAT)
      IL = 1
      IC = 1
      IR = 1
      IRUT = 1
      ZERO = CL * ZL + CR * ZR - ZC
      GL = GAINL / TRIM * CL
      GC = GAINC / TRIM
      GR = GAINR / TRIM * CR
      DO 20 I = 1, NS, TRIM
        SUM = 0
        DO 10 J = 1, TRIM
          SUM = SUM + GL * HL (IL) + GR * HR (IR) - GC * HC (IC)
          IL = IL + NCHRAW
          IC = IC + NCHRAW
          IR = IR + NCHRAW
   10   CONTINUE
        RUT (IRUT) = SUM - ZERO
   20 IRUT = IRUT + NCHRUT
      RETURN
```

END

SETSTM.FOR

```
$TITLE:'SETSTM'
$STORAGE:2
$NOFLOATCALLS
      SUBROUTINE SETSTM
*********************************************************************
*  This subroutine calculates coefficients for the state-transition
*  matrix used in the IRI quarter-car simulation.  It requires
*  MINV, a matrix inversion subroutine.
*********************************************************************
      REAL*4 A(4,4), A1(4,4), A2(4,4), MIN1(4), MIN2(4),
     &       K1, K2, MU, C
$INCLUDE:'SETCOM'
*
      DATA K1, K2, MU, C /653., 63.3, .15, 6./
      DT = DELTAX * SCLFDX / 80. * 3.6
*
*  Put 1/4 car model parameters into the A Matrix
*
      DO 60 J = 1, 4
        DO 50 I = 1, 4
          A(I,J) = 0
          A1(I,J) = 0
   50   STM(I,J) = 0
        A1(J,J) = 1.
   60 STM(J,J) = 1.
      A(1,2) = 1.
      A(3,4) = 1.
      A(2,1) = -K2
      A(2,2) = -C
      A(2,3) = K2
      A(2,4) = C
      A(4,1) = K2 / MU
      A(4,2) = C / MU
      A(4,3) = -(K1 + K2) / MU
   70 A(4,4) = -C / MU
*
*  CALCULATE STATE TRANSITION MATRIX - STM=EXP(A*DT) -
*  VIA A TAYLOR SERIES EXPANSION.
*
      ITER = 0
   80 ITER = ITER + 1
      ISTOP = 1
      DO 110 J = 1, 4
        DO 100 I = 1, 4
          A2(I,J) = 0
          DO 90 II = 1, 4
   90     A2(I,J) = A2(I,J) + A1(I,II) * A(II,J)
  100   CONTINUE
  110 CONTINUE
      DO 130 J = 1, 4
        DO 120 I = 1, 4
          A1(I,J) = A2(I,J) * DT / ITER
          IF (STM(I,J) + A1(I,J) .EQ. STM(I,J)) GO TO 120
          ISTOP = 0
```

```
         STM(I,J) = STM(I,J) + A1(I,J)
  120    CONTINUE
  130 CONTINUE
  140 CONTINUE
      IF (ISTOP .EQ. 0) GO TO 80
*
*  CALCULATE PARTICULAR RESPONSE MATRIX:  PRM=A**-1*(STM-I)*B
*
  150 CONTINUE
      DO 170 J = 1, 4
        DO 160 I = 1, 4
  160   A2(I,J) = A(I,J)
  170 CONTINUE
*
*  USE IBM MATRIX INVERSION SUBROUTINE (SSP LIBRARY)
*
      CALL MINV (A2, 4, DET, MIN1, MIN2)
      DO 190 I = 1, 4
        PRM(I) = -A2(I,4)
        DO 180 J = 1, 4
  180   PRM(I) = PRM(I) + A2(I,J) * STM(J,4)
  190 PRM(I) = PRM(I) * K1 / MU
  200 CONTINUE
      RETURN
      END
```

SETUP.FOR

```
$TITLE:'SETUP'
$STORAGE:2
$NOFLOATCALLS

      SUBROUTINE SETUPS

      CHARACTER*16 FN
      LOGICAL*2 IEXIST
      CHARACTER*1 DRIVE
      CHARACTER*3 EXT
      INTEGER*2 CP(7),PAGE,PAGMAX,ROW,COL
      CHARACTER*8 FMT(7)
$INCLUDE:'SETCOM'

C     PRINT SETUP TO SCREEN

      CALL SETCUR(2,0)
      WRITE(*,9000)
9000  FORMAT('CHAN    ID       UNITS   TYPE TRANSDUCER AMPLIFIER',
     1  ' OFFSET AT  AMPLIFIER   FULL'\)
      CALL SETCUR(3,0)
      WRITE(*,9010)
9010  FORMAT(' #',30X,'GAIN',5X,'GAIN(NOM) ZERO VOLTS GAIN(ACT)',
     1     '    SCALE'\)
      CALL SETCUR(4,0)
      WRITE(*,9020)
9020  FORMAT('****',1X,'********',2X,'********',2X,'**** **********',1X,
     1  '*********',' ********** *********   *******'\)

C     WRITE OUT CURRENT VALUES

      DO 100 I=1,9
      CALL SETCUR(I+5,0)
      J=I-1
      FSC=GAIN(I)*2048
      WRITE(*,9030)J,CHID(I),UNITS(I),XDUCT(I),XDUCGN(I),AMPGN(I),
     1  OFFS(I),AMPGA(I),FSC
9030  FORMAT(1X,I1,3X,A8,2X,A8,3X,I1,3X,F9.5,2X,F8.4,2X,F9.5,2X,F8.4,
     1  2X,F7.4\)
100   CONTINUE
      PAGE=1
      PAGMAX=1
      I=1
      J=1
      CP(1)=5
      CP(2)=15
      CP(3)=26
      CP(4)=30
      CP(5)=41
      CP(6)=51
      IMAX=9
      JMAX=6
200   ROW=5+I
      COL=CP(J)
```

```
      IF(J .EQ. 1)THEN
       CALL GETSTR(CHID(I),8,ROW,COL,IRET)
      ELSE IF (J .EQ. 2)THEN
       CALL GETSTR(UNITS(I),8,ROW,COL,IRET)
      ELSE IF (J .EQ. 3)THEN
       CALL GETI(XDUCT(I),0,1,ROW,COL,1,'(I1\)',IRET)
      ELSE IF(J .EQ. 4)THEN
       CALL GETR(XDUCGN(I),-10000.,10000.,ROW,COL,9,'(F9.5\)',IRET)
      ELSE IF(J .EQ. 5)THEN
       CALL GETR(AMPGN(I),0.001,2000.,ROW,COL,8,'(F8.4\)',IRET)
      ELSE IF (J .EQ. 6)THEN
       CALL GETR(OFFS(I),-10000.,10000.,ROW,COL,9,'(F9.5\)',IRET)
      ELSE
      ENDIF
      IF  (IRET .EQ. 9) GOTO 1000
      CALL RETPRO(IRET,J,I,JMAX,IMAX,PAGE,PAGMAX)
      GOTO 200
1000  CONTINUE
      CALL WRTSET
      RETURN
      END
```

SIGSUBS.FOR

```
$TITLE:'FUNCTION RAVE'
$NOFLOATCALLS
$STORAGE:2
**********************************************************************
      FUNCTION RAVE (ARRAY, NCH, NS)
**********************************************************************
*  Function to compute average value of signal in real*4 array.
*
*  <-- RAVE   real*4     average value of channel-1 in ARRAY.
*  --> ARRAY  real*4     2-D input array.  Channel 1 is processed.
*  --> NCH    integer*4  1st dimension (# of channels) for ARRAY.
*  --> NS     integer*4  2nd dimension (# of samples) for ARRAY.
*
$LARGE:ARRAY
      INTEGER*4 NCH, NS, I, J
      REAL*4 ARRAY(*)
      REAL*8 SUM8
      I = 1
      SUM8 = 0
      DO 10 J = 1, NS
        SUM8 = SUM8 + ARRAY (I)
   10 I = I + NCH
      RAVE = SUM8 / NS
      RETURN
      END
$PAGE
```

```
$TITLE:'SUBROUTINE DEBIAS'
************************************************************************
      SUBROUTINE DEBIAS (ARRAY, NCH, NS, BIAS)
***********************************************************************
*  Subroutine to remove bias from signal in real*4 array.
*
*  <-> ARRAY real*4     2-D array.  Channel 1 is processed.
*  --> NCH   integer*4  1st dimension (# of channels) for ARRAY.
*  --> NS    integer*4  2nd dimension (# of samples) for ARRAY.
*  --> BIAS  real*4     bias to be sutracted from channel-1 in ARRAY.
*
$LARGE: ARRAY
      INTEGER*4 NCH, NS, I, J
      REAL*4 ARRAY(*)
      I = 1
      DO 10 J = 1, NS
        ARRAY (I) = ARRAY (I) - BIAS
   10 I = I + NCH
      RETURN
      END
$PAGE
```

```
$TITLE:'FUNCTION IAVE'
************************************************************************
      FUNCTION IAVE (ARRAY, NCH, NS)
************************************************************************
*  Function to compute average value of signal in integer*2 array.
*
*  <-- IAVE  integer*2  average value of channel-1 in ARRAY.
*  --> ARRAY integer*2  2-D input array.  Channel 1 is processed.
*  --> NCH   integer*4  1st dimension (# of channels) for ARRAY.
*  --> NS    integer*4  2nd dimension (# of samples) for ARRAY.
*
$LARGE:ARRAY
      INTEGER*2 ARRAY(*), IAVE
      INTEGER*4 SUM4, NCH, NS, I, J
      I = 1
      SUM4 = 0
      DO 10 J = 1, NS
        SUM4 = SUM4 + ARRAY (I)
   10 I = I + NCH
      IAVE = SUM4 / NS
      RETURN
      END
$PAGE
```

```
$TITLE:'SUBROUTINE AVEVEL'
**********************************************************************
      SUBROUTINE AVEVEL (IBUF, NC1, NS, RBUF, NC2, TRIM, GAIN, BIAS)
**********************************************************************
*  Subroutine to average and decimate a (speed) signal.
*
*  --> IBUF  integer*2  2-D input array.  Channel 1 is processed.
*  --> NC1   integer*4  1st dimension (# of channels) for IBUF.
*  --> NS    integer*4  2nd dimension (# of samples) for IBUF.
*  <-- RBUF  real*4     2-D output array.  Channel 1 is processed.
*  --> NC2   integer*4  1st dimension (# of channels) for RBUF.
*  --> TRIM  integer*4  decimation ratio.  Every TRIM-th point is kept.
*  --> GAIN  real*4     gain for input data: engineering units/count.
*  --> BIAS  real*4     bias in input data.
*
$LARGE: IBUF,RBUF
      INTEGER*2 IBUF(*)
      INTEGER*4 NCNCR, NS, SUM, I, I1, I2, J, TRIM, NC1, NC2
      REAL*4 RBUF(*), BIAS, GAIN, GT
*
      GT = GAIN / TRIM
      I1 = 1
      I2 = 1
      DO 20 I = 1, NS, TRIM
        SUM = 0
        DO 10 J = 1, TRIM
          SUM = SUM + IBUF (I1)
   10   I1 = I1 + NC1
        RBUF (I2) = SUM * GT - BIAS
   20 I2 = I2 + NC2
      RETURN
      END
$PAGE
```

```
$TITLE:'SUBROUTINE PRFELV'
*************************************************************************
      SUBROUTINE PRFELV (BUF1, NC1, NS, BUF2, NC2, TRIM, DX, C,
     &                   ENDELV)
*************************************************************************
*  Subroutine to compute compressed elevation profile from slope.
*
*  --> BUF1   real*4     2-D input array.  Channel 1 is processed.
*  --> NC1    integer*4  1st dimension (# of channels) for BUF1.
*  --> NS     integer*4  2nd dimension (# of samples) for BUF1.
*  <-- BUF2   real*4     2-D output array.  Channel 1 is processed.
*  --> NC2    integer*4  1st dimension (# of channels) for BUF2.
*  --> TRIM   integer*4  decimation ratio.  Every TRIM-th point is kept.
*  --> DX     real*4     sample interval for BUF1
*  --> C      real*4     coefficient to add high-pass filtering to
*                        the integration.
*  <-> ENDELV real*4     as input, starting elevation at beginning of
*                        buffer.  as output, elevation at end of buffer.
*
$LARGE:BUF1, BUF2
      REAL*4 BUF1(*), BUF2(*), DX, C, ENDELV
      INTEGER*4 NS, I, I1, I2, NC1, NC2, TRIM, J
*
*  Integrate slope backwards.
*
      I1 = (NS - 1) * NC1 + 1
      I2 = (NS / TRIM - 1) * NC2 + 1
      DO 20 I = 1, NS, TRIM
        DO 10 J = 1, TRIM
          ENDELV = ENDELV * C + DX * BUF1 (I1)
   10   I1 = I1 - NC1
        BUF2 (I2) = ENDELV
   20 I2 = I2 - NC2
      RETURN
      END
$PAGE
```

```
$TITLE:'SUBROUTINE HIPASS'
*******************************************************************
        SUBROUTINE HIPASS (ARRAY, NCH, N1, N2, N3, N4, N5, MOVAV1,
     &                     MOVAV2)
*******************************************************************
*  This subroutine filters a signal with a hi-pass filter.  It is
*  based on the MTS subroutine HILOF, and customized for a variable
*  initilazation and ending section.
*
*  <-> ARRAY  real*4     2-D Input array.  Channel 1 is filtered.
*                        This array must be dimensioned to cover (N1 +
*                        (N2 + N3 + N4 + N5 + MOVAV1 + 1) samples.
*                        The input data should start at the second
*                        position and continue
*                        to the end of the array. The output
*                        starts at the first position, and continues
*                        to the N3-th position.
*  --> NCH    integer*4  1st dimension of ARRAY. (# of channels.)
*  --> N1-N5  integer*4  no. of samples in five contiguious regions of
*                        memory.
*  --> MOVAV1 integer*4  no. of points in moving average,
*  --> MOVAV2 integer*4  no. of points to center of moving average
*                        (MOVAV1 / 2),
*******************************************************************
$LARGE: ARRAY
      INTEGER*4 MOVAV1, MOVAV2, NCH, N1, N2, N3, N4, N5, I, I1, I2,
     &          M1, M2, N
      REAL*4 ARRAY (*), SCM1
*
*  Create artificial points to start the moving average if N1 > 0.
*
      IF (N1 .GT. 0) THEN
        N = MOVAV1 - N1
        IF (N .LT. N2 + N3 + N4) N = N2 + N3 + N4
        CALL LRSLOP (ARRAY ((N1 + 1) * NCH + 1), NCH, N, SLOPE)
        I1 = 1 + N1 * NCH
        I2 = I1 + NCH
        DO 10 I = 1, N1
          ARRAY (I1) = ARRAY (I2) - SLOPE * I
          I1 = I1 - NCH
   10   CONTINUE
      END IF
*
*  Create artificial points to finish the moving average if N5 > 0.
*
      IF (N5 .GT. 0) THEN
        N = MOVAV1 - MOVAV2 - N5
        IF (N .LT. N2 + N3 + N4) N = N2 + N3 + N4
        CALL LRSLOP (ARRAY ((1 + N1 + N2 + N3 + N4 - N) * NCH
     &               + 1), NCH, N, SLOPE)
        I2 = 1 + (N1 + N2 + N3 + N4) * NCH
        I1 = I2 + NCH
        DO 20 I = 1, N5
          ARRAY (I1) = ARRAY (I2) + SLOPE * I
          I1 = I1 + NCH
   20   CONTINUE
      END IF
```

```
*
*  Initialize moving average.
*
      ARRAY (1) = 0
      I1 = 1
      DO 40 I=1, MOVAV1
        I1 = I1 + NCH
        ARRAY (1) = ARRAY (1) + ARRAY (I1)
   40 CONTINUE
      ARRAY (1) = ARRAY (1) / MOVAV1
*
*  Filter signal.
*
      I1 = 1
      I2 = I1 + NCH
      SCM1 = 1. / MOVAV1
      M1 = MOVAV1 * NCH
      M2 = MOVAV2 * NCH
*
      DO 50 I = 2, N3
        ARRAY (I2) = ARRAY (I1) + SCM1 * (ARRAY (I2 + M1) -
     &               ARRAY (I2))
        ARRAY (I1) = ARRAY (I2 + M2) - ARRAY (I1)
        I1 = I2
        I2 = I2 + NCH
   50 CONTINUE
      ARRAY (I1) = ARRAY (I2 + M2) - ARRAY (I1)
      RETURN
      END
$PAGE
```

```
$TITLE:'SUBROUTINE LRSLOP'
**********************************************************************
      SUBROUTINE LRSLOP (ARRAY, NDIM, NSAMP, SLOPE)
**********************************************************************
*  Calculate slope of signal using a linear regression.
*
*  by Mike Sayers, last modified June 27, 1986.
*
*  --> ARRAY  real*4     2-D Input array.
*  --> NDIM   integer*4  1st dimension of ARRAY. (# of channels.)
*  --> NSAMP  integer*4  2nd dimension of ARRAY. (# of samples.)
*  <-- SLOPE  real*4     Slope of channel 1 in ARRAY as obtained by
*                        linear regression.
**********************************************************************
$LARGE:ARRAY
      DIMENSION ARRAY (*)
      INTEGER*4 NDIM, NSAMP, I, J
      REAL*8 SUMX, SUMXY, SUMY, SUMX2, X, Y
      SUMXY=0
      SUMX = 0
      SUMX2 = 0
      SUMY = 0
      SLOPE = 0
      IF (NSAMP .LT. 2) RETURN
*
      I = 1
      DO 10 J = 1, NSAMP
        X = J
        Y = ARRAY (I)
        SUMX = SUMX + X
        SUMXY = SUMXY + X * Y
        SUMX2 = SUMX2 + X * X
        SUMY = SUMY + Y
        I = I + NDIM
   10 CONTINUE
*
      SLOPE = (NSAMP * SUMXY - SUMX * SUMY) / (NSAMP * SUMX2 -
     &         SUMX * SUMX)
      RETURN
      END
```

STARTAD.FOR

```
$TITLE:'START A/D'
$STORAGE:2
$NOFLOATCALLS

      SUBROUTINE STARTAD(IITY,FF,BUFST,BUFT,BUFFCNT,MAXB,
     1       ADCURB,DONE,INDEX)

$INCLUDE:'BUFCOM'
$INCLUDE:'SETCOM'
$INCLUDE:'ADINSERT'
$INCLUDE:'IOPARMS'
      INTEGER*4 II,JJ,QQ
      INTEGER*2 QA(2),Q1(2),LOW,HIGH,AD(5),DM(5)
      EQUIVALENCE(JJ,QA),(QQ,Q1)
      CHARACTER*1 BP

C     INDICATE BUFFERS ARE EMPTY
      NBUF=15
      DO 10 I=1,NBUF
10    BUFST(I)=0

C     INITIALIZE FLAGS AND COUNTERS
      DONE=0
      ADCURB=0
      BUFFCNT=0
      II=0
      INDEX=0

C     CALCULATE BEGINNING OF BUFFERS SO THAT THERE ARE NO
C     PAGE OVERUNS

C     GET PHYSICAL ADDRESS OF IBUF-PUT OFFSET IN QQ
      CALL PHYSAD(IBUF(1),JJ)
      Q1(1)=QA(1)
      Q1(2)=0
      II=JJ

C     USE MAXIMUM BUFFER SIZE OF 16384 BYTES
      MBUFSIZ=16384

C     CALCULATE ACTUAL BYTES PER BUFFER
      NSAMP=MBUFSIZ/(NCHAN*2)
      IF (MOD(NCHAN,2) .EQ. 0)NSAMP=NSAMP-1
      BYTB=NSAMP*NCHAN*2

      J=0
      IF(QQ .EQ. 0) GOTO 100
      IF(QQ .GT. 49152)THEN
      QA(2)=QA(2)+1
      QQ=0
      ELSE IF (QQ .GT. 32768)THEN
      QQ=49152
      ELSE IF (QQ .GT. 16384)THEN
      QQ=32768
```

```
      ELSE
      QQ=16384
      ENDIF
100   QA(1)=Q1(1)
      INDEX=(JJ-II)/2+1

C     PUT PHYSICAL ADDRESS INTO BUFFER TABLE
      DO 200 I=1,NBUF
      BUFT(I)=(I-1)*16384 +JJ
200   CONTINUE

      DO 210 I=2,NBUF
210   BUFT(I)=BUFT(I)+1

C     RESET A/D CLOCK CIRCUITRY
      CALL IOUTB(2,CNTRL)
      M=8
      L=#00FF
      BYTB=BYTB-1

C     SETUP A/D AND DMA CONTROLLER

      AD(1)=1
      AD(2)=ADSTRT
      AD(3)=ADSTOP
      AD(4)=NCHAN
      AD(5)=0
      DM(1)=IAND(QA(1),L)
      DM(2)=ISHFTR(QA(1),M)
      DM(3)=IAND(BYTB,L)
      DM(4)=ISHFTR(BYTB,M)
      DM(5)=QA(2)
      CALL SETDMA(DM)
      CALL SETAD(AD)

C     START A/D -THEN WAIT .1 SEC
      CALL PWAIT(DTSTAT,CWAIT,0)
      CALL IOUTB(#7E,DTCOM)
      CALL TWAIT(.1)

C     SET UP COUNTERS BUT DON'T START COUNTER #1

C     DISARM ALL
      CALL IOUTB(#5F,TIMERC)
C     POINT TO COUNTER 1
      CALL IOUTB(1,TIMERC)

C     SET MODE
      LOW=IAND(IDMODE,L)
      HIGH=ISHFTR(IDMODE,M)
      CALL IOUTB(LOW,TIMERD)
      CALL IOUTB(HIGH,TIMERD)

C     SET DIVISOR
      LOW=IAND(IDIV,L)
      HIGH=ISHFTR(IDIV,M)
      CALL IOUTB(9,TIMERC)
      CALL IOUTB(LOW,TIMERD)
```

```
      CALL IOUTB(HIGH,TIMERD)

C     SET UP COUNTER #2 FOR 25.3868 KHZ FOR A/D CLOCK

C     POINT TO COUNTER 2 MODE REGISTER
      CALL IOUTB(2,TIMERC)
C     SET MODE=#0B22
      CALL IOUTB(#22,TIMERD)
      CALL IOUTB(#0B,TIMERD)
C     DIVIDE BY 47
      CALL IOUTB(#0A,TIMERC)
      CALL IOUTB(47,TIMERD)
      CALL IOUTB(0,TIMERD)

C     SET UP COUNTER 3 TO COUNT OUT2 BY NCHAN

C     POINT TO COUNTER 3 MODE REGISTER AND SET MODE=D3A5
      CALL IOUTB(3,TIMERC)
      CALL IOUTB(#A5,TIMERD)
      CALL IOUTB(#D3,TIMERD)

C     POINT TO COUNTER 3 LOAD REGISTER AND SET=NCHAN
      CALL IOUTB(#0B,TIMERC)
      CALL IOUTB(NCHAN,TIMERD)
      CALL IOUTB(0,TIMERD)

C     SET UP COUNTER 4 TO COUNT SAMPLES TO NSAMP

C     POINT TO COUNTER 4 MODE REGISTER AND SET=#1421
      CALL IOUTB(4,TIMERC)
      CALL IOUTB(#21,TIMERD)
      CALL IOUTB(#14,TIMERD)

C     POINT TO LOAD REGISTER AND SET=NSAMP*NCHAN
      I=NCHAN*NSAMP
      LOW=IAND(I,L)
      HIGH=ISHFTR(I,M)
      CALL IOUTB(#0C,TIMERC)
      CALL IOUTB(LOW,TIMERD)
      CALL IOUTB(HIGH,TIMERD)

C     SET AND START FILTER CLOCK(COUNTER 5)
      CALL FILCLK(FF)

C     LOAD AND ARM COUNTER 2,3,AND 4
      CALL IOUTB(#6E,TIMERC)

C     SETUP INTERRUPT SOFTWARE
      NBUF=NBUF-1
      CALL ADSET(ADCURB,BUFT,BUFST,NBUF,BYTB,MAXB,BUFFCNT,DONE)

C     WAIT FOR KEY
      CALL KCLEAR
300   I=IGKEY()
      IF(I .EQ. 0)GOTO 300

C     IF THIS IS A BOUNCE TEST WAIT 10 SECS THEN BEEP THEN START
      IF(IITY .EQ. 1)THEN
```

```
      CALL TWAIT(10.)
      BP=CHAR(7)
      WRITE(*,'(A\)')BP
      ENDIF
C     START A/D
      CALL IOUTB(7,CNTRL)
      CALL IOUTB(6,CNTRL)
      CALL IOUTB(3,CNTRL)
C     ENABLE INTERRUPTS
      CALL IOUTB(1,CNTRL)
      CALL IOUTB(0,INTE)
C     START COUNTER 1
      CALL IOUTB(#61,TIMERC)
1000  RETURN
      END
```

TEST.FOR

```
$TITLE:'TEST'
$STORAGE:2
$NOFLOATCALLS

      SUBROUTINE TEST(IITY)

C     IITY=TEST TYPE
C     0=NORMAL TEST
C     1=BOUNCE TEST
$INCLUDE:'BUFCOM'
$INCLUDE:'STATCOM'
$INCLUDE:'SETCOM'
$INCLUDE:'ADINSERT'
$INCLUDE:'IOPARMS'
$INCLUDE:'HANDLES'
      INTEGER*2 PAGE,PAGMAX,CP(12),ROW,COL,ISTAT(3)
      INTEGER*2 WRB,WRBL,WRTCNT
      INTEGER*4 JJ,JL,JK,ADDRF(15),ADDRL(15),IO,IP,BSTRT(15),NRAWFW
      INTEGER*4 BYNEED,BYAV,MAXSAMP,BBB,CLUSA,CLUST,SECTOR,EXBUF
      LOGICAL*2 IEXIST,WRT
      CHARACTER*8 FN
      CHARACTER*1 DR
      CHARACTER*3 EXT
      CALL CLRSCR
C     SET PARAMETERS FOR PROCESSING
      PASSA=0
      TRIM=10
      LNGWAV=50.
      SCLFA=9.805
      SCLFDX=.3048
      SCLFH=.0254
      SCLFV=.44703
      SCLFRI=5280.
      H1LAT=27.0625
      H3LAT=27.6875
      H4LAT=0.
      H5LAT=0.
      TSTTYP=IITY

C     CHECK TO SEE IF THERE IS A TAPE LOADED
10    IF(TINIT .EQ. 0)THEN
      CALL SETCUR(10,0)
      WRITE(*,'(A\)')'THERE IS NO TAPE LOADED --PLEASE LOAD A TAPE'
      CALL WAITKY
      CALL LOADT
      ENDIF
      CALL CLRSCR
      CALL FNMAKE(DR,FN,EXT,TFILE,1)
      IF(IITY .EQ. 0)THEN
      EXT='DTA'
      ELSE
      EXT='BNC'
      ENDIF
      CALL FNMAKE(DR,FN,EXT,TFILE,0)
```

```
C      GET DATE AND TIME
       CALL GDATE(IYR,IM,IDAY)
       CALL GTIME(IH,IMIN,ISEC)

C      BLANK OUT COMMENT
       CALL BLANK(CMT,64)

C      WRITE TEST DISPLAY
       CALL TSTDIS
       PAGE=1
       PAGMAX=1
       I=1
       IMAX=6
       J=1
       JMAX=2
100    CALL CLRLIN(23)
       IF( I .EQ. 1)THEN
       CALL FNMAKE(DR,FN,EXT,TFILE,1)
       CALL GFILE(DR,FN,EXT,IEXIST,5,12,IRET)
        IF(IEXIST)THEN
         CALL INERROR('FILE ALREADY EXISTS',19)
         GOTO 100
        ELSE
         CALL FNMAKE(DR,FN,EXT,TFILE,0)
        ENDIF
       ELSE IF (I .EQ.2)THEN
        CALL GETSTR(CMT,64,6,9,IRET)
       ELSE IF (I .EQ.3)THEN
        IF (J .EQ. 1)THEN
        CALL GETSTR(ROUTE,16,8,7,IRET)
        ELSE
        CALL GETI(TSTSPD,10,55,8,52,2,'(I2\)',IRET)
        ENDIF
       ELSE IF (I .EQ.4)THEN
        IF(J .EQ. 1)THEN
        CALL GETSTR(DIRECT,8,9,10,IRET)
        ELSE
        CALL GETR(MAXLEN,.1,20.,9,60,4,'(F4.1\)',IRET)
        ENDIF
       ELSE IF (I .EQ.5)THEN
        IF(J .EQ. 1)THEN
        CALL GETSTR(LANE,12,10,5,IRET)
        ELSE
        CALL GETR(SAMP,.01,4.,10,57,4,'(F4.2\)',IRET)
        ENDIF
       ELSE IF (I .EQ.6)THEN
        IF(J .EQ. 1)THEN
        CALL GETSTR(SURF,16,11,13,IRET)
        ELSE
        CALL GETSTR(OPER,16,11,49,IRET)
        ENDIF
       ELSE
       ENDIF
       IF (IRET .EQ. 9)GOTO 300
       CALL RETPRO(IRET,J,I,JMAX,IMAX,PAGE,PAGMAX)
       IF( I .LT. 3 .AND. J .EQ. 2)THEN
        I=I+1
```

```
      J=1
      ENDIF
      GOTO 100

300   CONTINUE

      IF(IITY .EQ. 0)THEN

C     NORMAL TEST--DO DISTANCE BASED SAMPLING
       D=12.0/SAMP
       IDIV=NINT(D/XDUCGN(9))
       IDMODE=#0221

C      SET FILTER FREQUENCY TO 1/3 OF NOMINAL SAMPLING FREQUENCY
       FF=SAMP*88.0/60.0*TSTSPD/3.0

C      COMPUTE DELTAX IN METERS
       DELTAX=IDIV*XDUCGN(9)/12.0
       DXTRIM=DELTAX*TRIM
       T1=DELTAX

C      SET QUARTER CAR MATRIX
       CALL SETSTM

C     CALCULATE MAXIMUM TEST LENGTH
       MAXSAMP=5280.*MAXLEN/DELTAX
       EXBUF=0
       NRSAMP=(MAXSAMP-1)/TRIM
       NRUTFW=NRSAMP*NCHRUT
       NRAWFW=NCHAN*MAXSAMP/2 +2+.5
       MAXBUF=NRAWFW+NRUTFW
       IF(MAXBUF .GT. MXBFSZ)EXBUF=16

      ELSE
C     TIME BASED SAMPLING
       EXBUF=0
       MAXSAMP=4096
       F1=4096./20.
       FF=F1/3
       D=2.0*1.193182E6/F1
       IDIV=NINT(D)
       IDMODE=#0B21
       T1=IDIV*.4190477E-6
      ENDIF

      NPTS=16384/(NCHAN*2)
      IF(MOD(NCHAN,2)    .EQ. 0)NPTS=NPTS-1
      M=NPTS*NCHAN*2-1
      BYTES=M+1

C     CALCULATE THE MAXIMUM NUMBER OF BUFFERS
      MAXB=MAXSAMP/NPTS
      IF( MOD(MAXSAMP,NPTS) .NE. 0)MAXB=MAXB+1

C     CALCULATE THE # OF DISK BYTES NEEDED
      BYNEED=16384*(MAXB+EXBUF)+2048

C     CHECK FOR ROOM ON THE TAPE
```

```
      DR=TFILE(1:1)
      IDR=ICHAR(DR)-64
305   CALL DFREE(IDR,CLUSA,CLUST,BBB,SECTOR)
      BYAV=1.0*CLUSA*BBB*SECTOR
      IF (BYAV .LT. BYNEED)THEN
      CALL SETCUR(16,0)
      WRITE(*,9010)MAXLEN,DR
9010  FORMAT('A TEST OF ',F4.1,' MILES WILL NOT FIT ON DRIVE ',A1\)
      CALL SETCUR(17,0)
      WRITE(*,9020)
9020  FORMAT('DO YOU WANT TO GO TO THE NEXT DRIVE? '\)
      CALL GCUR(IR,IC)
      IANS=1
      CALL YESNO(IANS,IR,IC,IRET)
      IF (IANS .EQ. 0)GOTO 10
       IF(DR .EQ. 'F')THEN
        CALL SETCUR(18,0)
        WRITE(*,'(A\)')' THE TAPE IS FULL- CHANGE TAPES'
        RETURN
       ELSE
        IDR=IDR+1
        DR=CHAR(IDR+64)
        TFILE(1:1)=DR
        GOTO 305
       ENDIF
      ENDIF

      WRT=.FALSE.
      WRB=1
      WRBL=1
      WRTCNT=0

      CALL SETCUR(19,0)
      WRITE(*,'(A\)')'HIT ANY KEY TO START'
      CALL CLRLIN(24)
      CALL STARTAD(IITY,FF,BUFST,BUFT,BUFFCNT,MAXB,ADCURB,DONE,INDEX)

C     CALULATE ADDRESSES FOR BYTE MOVES
      DO 310 I=1,15
      BSTRT(I)=INDEX+(I-1)*8192+1
      IF(I .EQ. 1)BSTRT(I)=BSTRT(I)-1
      CALL IVARPT(IBUF(BSTRT(I)),ADDRF(I))
      ADDRF(I)=ADDRF(I)-1
      CALL IVARPT(IBUF(BSTRT(I)+NCHAN*NPTS),ADDRL(I))
      ADDRL(I)=ADDRL(I)-1
310   CONTINUE

      CALL SETCUR(19,15)
      WRITE(*,'(A\)')'STOP '
      CALL SETCUR(20,0)
      CALL KCLEAR
      IF(IITY .EQ. 1)THEN
      WRITE(*,'(A\)')'ELAPSED TIME='
      CALL GCUR(IR,IC)
      ELSE
      WRITE(*,'(A\)')'ELAPSED DISTANCE'
      CALL GCUR(IR,IC)
      ENDIF
```

```
400    IF (DONE .NE. 0)GOTO 600

C      CHECK FOR WRITE
       IF (WRT)THEN
        IF(BUFST(WRB) .EQ. -1)THEN
         J=IPEEKB(ADDRF(WRB))
         CALL IPOKEB(J,ADDRL(WRBL))
         CALL HWRITE(HANDLE,IBUF(BSTRT(WRBL)),BYTES,RBYTES,IER)
         BUFST(WRBL)=0
         WRTCNT=WRTCNT+1
         IF(WRTCNT .EQ. 1)THEN
          BSTRT(1)=BSTRT(1)+1
          CALL IVARPT(IBUF(BSTRT(1)),ADDRF(1))
          ADDRF(1)=ADDRF(1)-1
          CALL IVARPT(IBUF(BSTRT(1)+NCHAN*NPTS),ADDRL(1))
          ADDRL(1)=ADDRL(1)-1
          BUFT(1)=BUFT(1)+1
         ENDIF
         WRBL=WRB
         WRB=WRB+1
         IF(WRB .GT. 15)WRB=1
        ENDIF
       ELSE
        IF(BUFST(WRB) .EQ. -1)THEN
         WRB=WRB+1
         WRT=.TRUE.
         CALL ADDNUL(TFILE,16)
         CALL HCREAT(TFILE,HANDLE,IER)
         CALL SUBNUL(TFILE,16)
         IF(IER .NE. 0)GOTO 5000

C        RECORD SETUP
         CALL HWRITE(HANDLE,SET,2048,RBYTES,IER)
        ENDIF
       ENDIF

       CALL SETCUR(IR,IC)
       I=BUFFCNT
       J=0
       CALL IOUTB(0,12)
       J=INPB(3)
       K=INPB(3)
       J=(M-IOR(J,ISHFTL(K,8)))/NCHAN/2
       T=T1*(J+I*NPTS)
       WRITE(*,'(F11.3\)')T
       IF(IGKEY() .EQ. 0)GOTO 400
600    I=IOR(INPB(#21),4)
       CALL IOUTB(0,INTD)
       CALL DTCLEAR
       CALL SETCUR(21,0)
       IF (DONE .EQ. 0)THEN
       WRITE(*,9100)
9100   FORMAT('TEST TERMINATED BY OPERATOR'\)
       ELSEIF ( DONE .EQ. -1)THEN
       WRITE(*,9110)
9110   FORMAT('MAXIMUM TEST LENGTH REACHED'\)
       ELSE
       WRITE(*,9120)
```

```
9120  FORMAT('TEST TERMINATED BY BUFFER OVERFLOW POSSIBILITY'\)
      ENDIF
      CALL SETCUR(22,0)
      WRITE(*,9000)
9000  FORMAT('RECORDING DATA'\)

      IF ( DONE .EQ. 0) THEN
       J=0
       CALL IOUTB(0,12)
       J=INPB(3)
       K=INPB(3)
       J=(M-IOR(J,ISHFTL(K,8)))/NCHAN/2
       PASSA=BUFFCNT*NPTS+J
       IF(J .NE. 0)BUFFCNT=BUFFCNT+1
      ELSE
       PASSA=BUFFCNT*NPTS
      ENDIF
      IF(BUFFCNT .EQ. 0)GOTO 900

C     RECORD REST OF DATA
775   IF(WRTCNT .EQ. BUFFCNT)GOTO 800
      IF( .NOT. WRT )THEN
        WRT=.TRUE.
        CALL ADDNUL(TFILE,16)
        CALL HCREAT(TFILE,HANDLE,IER)
        CALL SUBNUL(TFILE,16)
        IF(IER .NE. 0)GOTO 5000

C       RECORD SETUP
        CALL HWRITE(HANDLE,SET,2048,RBYTES,IER)
      ENDIF
      J=IPEEKB(ADDRF(WRB))
      CALL IPOKEB(J,ADDRL(WRBL))
      CALL HWRITE(HANDLE,IBUF(BSTRT(WRBL)),BYTES,RBYTES,IER)
      BUFST(WRBL)=0
      WRTCNT=WRTCNT+1
      WRBL=WRB
      WRB=WRB+1
      IF(WRB .GT. 15)WRB=1
      GOTO 775
800   CONTINUE
      PASSA=PASSA-1

C     CHECK TO SEE IF EXTRA ROOM IS NEEDED
       EXBUF=0
       NRSAMP=(PASSA-1)/TRIM
       NRUTFW=NRSAMP*NCHRUT
       NRAWFW=NCHAN*PASSA/2 +2+.5
       MAXBUF=NRAWFW+NRUTFW
       IF(MAXBUF .GT. MXBFSZ)EXBUF=16

      IF(EXBUF .NE. 0)THEN
C     WRITE 16 BUFFERS TO END
      BYTES=16384
      DO 840 I=1,16
      CALL HWRITE(HANDLE,IBUF(1),BYTES,RBYTES,IER)
840   CONTINUE
      ENDIF
```

```
C     POSITION TAPE BACK TO BEGINNING
      METHOD=0
      IO=0
      CALL HPOS(HANDLE,METHOD,IO,IP,IER)

C     RECORD SETUP
      CALL HWRITE(HANDLE,SET,2048,RBYTES,IER)

C     CLOSE FILE
      CALL HCLOSE(HANDLE,IER)
C     FLUSH DIRECTORY BUFFERS
      CALL TAPE(3,4,ISTAT)

900   IF(IITY .EQ. 1)RETURN
      CALL WAITKY
      RETURN
5000  CALL INERROR('FILE ERROR',10)
      CALL WAITKY
      RETURN
      END
```

TSTDIS.FOR

```
$TITLE:'TEST DISPLAY'
$STORAGE:2
$NOFLOATCALLS

      SUBROUTINE TSTDIS
***********************************************************************
*  This subroutine displays information about the data in a file
*  using infromation from the SETCOM header block.
*  If PASSA < 1, file is taken from TFILE and the bottom half of the
*  screen is left blank.  If PASSA > 0, then it is an existing data
*  and extra information is whown in the bottom 1/2 of the screen.

$INCLUDE:'STATCOM'
$INCLUDE:'SETCOM'
      CHARACTER*1 DR
      CHARACTER*8 FN
      CHARACTER*3 EXT
      CHARACTER*10 STR1, STR2, STR3
      INTEGER*2 IPTR(8)

      CALL WRTSCR('TSTSCR.     ')
C     DECODE FILENAME
      IF (PASSA .LE. 0) THEN
        CALL FNMAKE(DR,FN,EXT,TFILE,1)
      ELSE
        CALL FNMAKE(DR,FN,EXT,PFILE,1)
      END IF
      CALL SETCUR(5,10)
      WRITE(*,8900)DR,FN,EXT
8900  FORMAT(A1,':',A8,'.',A3\)

C     WRITE DATE AND TIME

      CALL SETCUR(1,5)
      WRITE(*,9000)IM,IDAY,IYR
9000  FORMAT(I2,'-',I2,'-',I4\)
      CALL SETCUR(2,5)
      WRITE(*,9010)IH,IMIN,ISEC
9010  FORMAT(I2,':',I2,':',I2\)

C     WRITE CONFIGURATION
      CALL SETCUR(3,15)
      WRITE(*,'(A\)')TSTCON

C     WRITE COMMENT
      CALL SETCUR(6,9)
      WRITE(*,'(A\)')CMT

C     WRITE ROUTE
      CALL SETCUR(8,7)
      WRITE(*,'(A\)')ROUTE

C     WRITE DIRECTION
      CALL SETCUR(9,10)
```

```
      WRITE(*,'(A\)')DIRECT

C     WRITE LANE
      CALL SETCUR(10,5)
      WRITE(*,'(A\)')LANE

C     WRITE SURFACE
      CALL SETCUR(11,13)
      WRITE(*,'(A\)')SURF

C     WRITE TEST SPEED
      CALL SETCUR(8,52)
      WRITE(*,'(I2\)')TSTSPD

C     WRITE MAXIMUM TEST LENGTH
      CALL SETCUR(9,60)
      WRITE(*,'(F4.1\)')MAXLEN

C     WRITE SAMPLES PER FOOT
      CALL SETCUR(10,57)
      WRITE(*,'(F4.2\)')SAMP

C     WRITE OPERATOR
      CALL SETCUR(11,49)
      WRITE(*,'(A\)')OPER

*   Quit now if PASSA < 1 (since this is during test setup)

      IF (PASSA .LE. 0) RETURN

*   Write status of file (raw, bounce, etc.)

      XLEN = PASSA * DELTAX
      L1 = 10
      CALL STRX (XLEN, STR1, L1)

      XLEN = PASSA * IDIV * .4190477E-06
      L2 = 10
      CALL STRX (XLEN, STR2, L2)

      XLEN = NPSTOT * DELTAX
      L3 = 10
      CALL STRX (XLEN, STR3, L3)

      CALL SETCUR (13,0)
      IF (TSTTYP .EQ. 0) THEN
        WRITE (*,'(A,A,A\)')
     &    'Road data that have not been checked or processed.',
     &    '  Length = ', STR1(:L1)
      ELSE IF (TSTTYP .EQ. 1) THEN
        WRITE (*,'(A,A,A\)')
     &    'Raw data from bounce test.',
     &    '  Time = ', STR2(:L2)
      ELSE IF (TSTTYP .EQ. 2) THEN
        WRITE (*,'(A,A,A\)')
     &    'Processed profile/rut-depth/roughness data.',
     &    '  Length = ', STR3(:L3)
      ELSE IF (TSTTYP .EQ. 3) THEN
```

```
        WRITE (*,'(A,A,A\)')
     &    'Raw data from road test that have been checked.',
     &    '  Length = ', STR1(:L1)
      ELSE IF (TSTTYP .EQ. 4) THEN
        WRITE (*,'(A,A,A\)')
     &    'Raw data that cannot be processed due to low speed.',
     &    '  Length = ', STR1(:L1)
      ELSE IF (TSTTYP .EQ. 5) THEN
        WRITE (*,'(A,A,A\)')
     &    'Raw data from bounce test that have been checked.',
     &    '  Time = ', STR2(:L2)
      ELSE IF (TSTTYP .EQ. 6) THEN
        WRITE (*,'(A,A,A\)')
     &    'Processed data from bounce test.',
     &    '  Time = ', STR2(:L2)
      ELSE IF (TSTTYP .EQ. 7) THEN
        WRITE (*,'(A\)')
     &   'This file was damaged during data processing and is ruined.'
      END IF

*  If that data were checked, print the findings.

      IF (TSTTYP .GT. 1) THEN
        L = ADSTRT
        DO 5 ICH = 1, NCHAN
          IPTR (ICH) = L + 1
          L = L + 1
          IF (L .GT. 7) L = 0
    5   CONTINUE

        DO 10 ICH = 1,NCHAN
          CALL SETCUR (13 + ICH, 0)
          IF (NSAT (ICH) .EQ. 0) THEN
            WRITE (*,'(A,'' SIGNAL WAS OK.''\)') CHID (IPTR(ICH))
          ELSE
            WRITE (*, 9050)  CHID (IPTR(ICH)), NSAT (ICH),
     &           LSAT (ICH) * DELTAX
9050        FORMAT (A,' WAS QUESTIONABLE',I5,' TIME(S), 1ST AT X=',
     &                      F9.2\)
          END IF
   10     CONTINUE
        IF (ICHV .GT. 0) THEN
          CALL SETCUR (22,0)
          WRITE (*,9060) VELMIN,VELMAX,UNITS(3)
 9060     FORMAT ('SPEED RANGE DURING TEST: ',F6.2,' TO',F6.2,1X,A\)
        END IF
      END IF

**********************************************
*  Set some of the things in common that were not set in early versions
*  of the test program.  (This code should be removed someday.)
      IF (TSTTYP .EQ. 1 .OR. TSTTYP .EQ. 5 .OR. TSTTYP .EQ. 6) THEN
        CHID (10) = 'TIME'
        CHID (11) = 'TOT AXLE'
        UNITS (10) = 'SECONDS'
        UNITS (11) = 'IN'
        SCLFRI = 1.
      ELSE
```

```
          SCLFRI =  5280.
          CHID (10) = 'DISTANCE'
          CHID (11) = 'IRI'
          UNITS (10) = 'FEET'
          UNITS (11) = 'IN/MI'
        END IF
        H1LAT = 1.
        H2LAT = 1.
        H4LAT = 1.
        H5LAT = 1.
*  <End patch>
***********************************************

        RETURN
        END
```

UNLOADTP.FOR

```
$STORAGE:2
$NOFLOATCALLS

      SUBROUTINE UNLDT

$INCLUDE:'BUFCOM'
$INCLUDE:'STATCOM'
$INCLUDE:'SETCOM'
      INTEGER*2 ISTAT(3)
      CHARACTER*16 FN
      FN='D:NAME.VOL    '

C     MAKE SURE TAPE IS TO BE UNLOADED
      CALL SETCUR(0,0)

C     IF NO TAPE IS LOADED - EXIT
      IF( TINIT.EQ. 0)THEN
      WRITE(*,'(A\)')'NO TAPE IS LOADED'
      GOTO 1000
      ENDIF

      WRITE(*,'(A\)')'ARE YOU SURE YOU WANT THE TAPE UNLOADED?'
      CALL SETCUR(0,41)
      I=0
      CALL YESNO(I,0,41,IRET)
      IF( I .EQ. 0)GOTO 1000

C     UPDATE VOLUME INFO

      LFILE=TFILE
      I=IYR-1900
      WRITE(TLDATE,9000)IM,IDAY,I
9000  FORMAT(I2,'-',I2,'-',I2)
      WRITE(TLTIME,9010)IH,IMIN,ISEC
9010  FORMAT(I2,':',I2,':',I2)

C     WRITE NEW INFO TO FILE D:NAME.VOL
      OPEN(9,FILE=FN)
      WRITE(9,8000)TVOL,(IBUF(I),I=1,100)
8000  FORMAT(A56,100I7)
      CLOSE(9)
C     FLUSH BUFFERS
      CALL TAPE(3,4,ISTAT)

C     UNLOAD TAPE
      CALL TAPE(12,4,ISTAT)
      TINIT=0
1000  CALL WAITKY
      RETURN
      END
```

```
C
C       READ FILE AND WRITE A SCREEN
C       FIRST LINE  NLINES=NUMBER OF LINES
C       SUBSEQUENT LINES = ROW,COL,STRING
C
```

WRTSCR.FOR

```
$TITLE:'WRITE DISPLAY'
$STORAGE:2
$NOFLOATCALLS

      SUBROUTINE WRTSCR(FNAME)

      CHARACTER*12 FNAME
      CHARACTER*80 STRING
      CHARACTER*1 STR(80)
      INTEGER*2 ROW,COL
      EQUIVALENCE(STRING,STR)

C     CLEAR THE SCREEN
      CALL CLRSCR

C     OPEN THE FILE AND GET THE NUMBER OF STRINGS

      OPEN(9,FILE=FNAME)
      READ(9,'(I4)')NUMSTR

C     READ THE STRINGS AND WRITE THEM TO THE SCREEN

      DO 100 I=1,NUMSTR
      READ(9,9000)ROW,COL,STRING
9000  FORMAT(I4,I4,A80)
      CALL HOWLNG(STRING,80,NCHAR)

      DO 50 J=1,NCHAR
      CALL SETCUR(ROW,COL)
      CALL PCHAR(STR(J),7,1)
50    COL=COL+1
100   CONTINUE
      RETURN
      END
```

REFERENCES

[1] T. D. Gillespie, M.W. Sayers, and M. R. Hagan, "Methodology for Road Roughness Profiling and Rut Depth Measurement." FHWA Report FHWA/RD-87/042, July 1987.

[2] M. W. Sayers, T. D. Gillespie, and M. R. Hagan, "User's Manual for the UMTRI/FHWA Road Profiling (PRORUT) System." FHWA Report FHWA/RD-87/043, July 1987.

[3] M. W. Sayers, and T. D. Gillespie, "The Ann Arbor Road Profilometer Meeting." FHWA Report No. FHWA/RD-86/100, July 1986, 226 pp.

[4] J. D. King and S. A. Cerwin, "System for Inventorying Road Surface Topography (SIRST)." Southwest Research Institute, Report No. FHWA/RD-82/062, August 1982, 269 pp.

[5] J. D. Campbell and M. W. Sayers, "An Infrared Distance Sensor, Analysis and Test Results." The University of Michigan Transportation Research Institute, Report No. UMTRI-84-14, March 1984, 114 pp.

[6] "Precision Non-Contact Measurement is Simpler than You Think," Available from Selective Electronic Inc., P.O. Box 100, Valdese, N.C. 28690.

[7] E. Spangler and W. Kelly, "GMR Road Profilometer—A Method for Measuring Road Profile." HRR121, Highway Research Board, 1966.

[8] M. W. Sayers, T. D. Gillespie, and W. D. O. Paterson, "Guidelines for the Conduct and Calibration of Road Roughness Measurements." Technical report No. 46, The World Bank, Washington D.C., January 1986, 87 pp.

*U.S. GOVERNMENT PRINTING OFFICE:1988-516-018:80276

FEDERALLY COORDINATED PROGRAM (FCP) OF HIGHWAY RESEARCH, DEVELOPMENT, AND TECHNOLOGY

The Offices of Research, Development, and Technology (RD&T) of the Federal Highway Administration (FHWA) are responsible for a broad research, development, and technology transfer program. This program is accomplished using numerous methods of funding and management. The efforts include work done in-house by RD&T staff, contracts using administrative funds, and a Federal-aid program conducted by or through State highway or transportation agencies, which include the Highway Planning and Research (HP&R) program, the National Cooperative Highway Research Program (NCHRP) managed by the Transportation Research Board, and the one-half of one percent training program conducted by the National Highway Institute.

The FCP is a carefully selected group of projects, separated into broad categories, formulated to use research, development, and technology transfer resources to obtain solutions to urgent national highway problems.

The diagonal double stripe on the cover of this report represents a highway. It is color-coded to identify the FCP category to which the report's subject pertains. A red stripe indicates category 1, dark blue for category 2, light blue for category 3, brown for category 4, gray for category 5, and green for category 9.

FCP Category Descriptions

1. **Highway Design and Operation for Safety**
   Safety RD&T addresses problems associated with the responsibilities of the FHWA under the Highway Safety Act. It includes investigation of appropriate design standards, roadside hardware, traffic control devices, and collection or analysis of physical and scientific data for the formulation of improved safety regulations to better protect all motorists, bicycles, and pedestrians.

2. **Traffic Control and Management**
   Traffic RD&T is concerned with increasing the operational efficiency of existing highways by advancing technology and balancing the demand-capacity relationship through traffic management techniques such as bus and carpool preferential treatment, coordinated signal timing, motorist information, and rerouting of traffic.

3. **Highway Operations**
   This category addresses preserving the Nation's highways, natural resources, and community attributes. It includes activities in physical maintenance, traffic services for maintenance zoning, management of human resources and equipment, and identification of highway elements that affect the quality of the human environment. The goals of projects within this category are to maximize operational efficiency and safety to the traveling public while conserving resources and reducing adverse highway and traffic impacts through protections and enhancement of environmental features.

4. **Pavement Design, Construction, and Management**
   Pavement RD&T is concerned with pavement design and rehabilititation methods and procedures, construction technology, recycled highway materials, improved pavement binders, and improved pavement management. The goals will emphasize improvements to highway performance over the network's life cycle, thus extending maintenance-free operation and maximizing benefits. Specific areas of effort will include material characterizations, pavement damage predictions, methods to minimize local pavement defects, quality control specifications, long-term pavement monitoring, and life cycle cost analyses.

5. **Structural Design and Hydraulics**
   Structural RD&T is concerned with furthering the latest technological advances in structural and hydraulic designs, fabrication processes, and construction techniques to provide safe, efficient highway structures at reasonable costs. This category deals with bridge superstructures, earth structures, foundations, culverts, river mechanics, and hydraulics. In addition, it includes material aspects of structures (metal and concrete) along with their protection from corrosive or degrading environments.

9. **RD&T Management and Coordination**
   Activities in this category include fundamental work for new concepts and system characterization before the investigation reaches a point where it is incorporated within other categories of the FCP. Concepts on the feasibility of new technology for highway safety are included in this category. RD&T reports not within other FCP projects will be published as Category 9 projects.

B68/5

HNR-20/3-88(120)QE