

## N O T I C E

THIS DOCUMENT HAS BEEN REPRODUCED FROM  
MICROFICHE. ALTHOUGH IT IS RECOGNIZED THAT  
CERTAIN PORTIONS ARE ILLEGIBLE, IT IS BEING RELEASED  
IN THE INTEREST OF MAKING AVAILABLE AS MUCH  
INFORMATION AS POSSIBLE

## N O T I C E

THIS DOCUMENT HAS BEEN REPRODUCED FROM  
MICROFICHE. ALTHOUGH IT IS RECOGNIZED THAT  
CERTAIN PORTIONS ARE ILLEGIBLE, IT IS BEING RELEASED  
IN THE INTEREST OF MAKING AVAILABLE AS MUCH  
INFORMATION AS POSSIBLE



(Ans-A-Ca-1036) PUBLIC AND LIBRARY  
INSTITUTION FOR THE ADVANCEMENT OF KNOWLEDGE,  
AND THE PROMOTION OF POLYTECHNIC INSTITUTE,  
TROY, N. Y., U. S. A. MARCH 17, 1911. SGCL 151

SGCL 151-1036

SGCL 151  
23072

Rensselaer Polytechnic Institute  
Troy, New York 12181

RPI TECHNICAL REPORT MP-76

ELECTRONIC AND SOFTWARE SUBSYSTEMS  
FOR AN AUTONOMOUS ROVING VEHICLE

by

Graham A. Doig

A Study Supported by the

NATIONAL AERONAUTICS AND SPACE ADMINISTRATION

under

Grant NSG-7369

and by the

JET PROPULSION LABORATORY

under

Contract 954880

School of Engineering  
Rensselaer Polytechnic Institute  
Troy, New York

October 1980



RPI TECHNICAL REPORT MP-76

ELECTRONIC AND SOFTWARE SUBSYSTEMS  
FOR AN AUTONOMOUS ROVING VEHICLE

by

Graham A. Doig

A Study Supported by the

NATIONAL AERONAUTICS AND SPACE ADMINISTRATION

under

Grant NSG-7369

and by the

JET PROPULSION LABORATORY

under

Contract 954880

School of Engineering  
Rensselaer Polytechnic Institute  
Troy, New York

October 1980



ELECTRONIC AND SOFTWARE SUBSYSTEMS  
FOR AN AUTONOMOUS ROVING VEHICLE

by

Graham A. Doig

A Project Submitted to the Graduate  
Faculty of Rensselaer Polytechnic Institute  
in Partial Fulfillment of the  
Requirements for the Degree of  
MASTER OF ENGINEERING

Major Subject: Electrical Engineering

Approved:

Dean K. Frederick  
Dean K. Frederick, Project Advisor

Rensselaer Polytechnic Institute  
Troy, New York

December 1980

## CONTENTS

	page
LIST OF FIGURES . . . . .	v
LIST OF TABLES . . . . .	vi
LIST OF ACRONYMS . . . . .	vii
ABSTRACT . . . . .	ix
1. INTRODUCTION . . . . .	1
2. SIMPLIFIED OVERVIEW. . . . .	4
3. MICROPROCESSOR CONTROL OF THE ROVER. . . . .	7
3.1 Program Organization. . . . .	11
3.2 Command Decoding. . . . .	12
3.3 Wheel Speed Control . . . . .	16
3.4 Display of Vehicle Parameters . . . . .	20
3.5 Vehicle Data Acquisition. . . . .	20
4. LASER MAST . . . . .	23
4.1 Electronic Mast Controller Capabilities . . . . .	24
4.2 Laser Mast Commands . . . . .	28
4.3 Laser Electronics and Power Supply. . . . .	29
4.4 Laser Data Description. . . . .	29
5. TELEMETRY SYSTEM . . . . .	32
5.1 Data Format . . . . .	32
5.2 Vehicle and Laser Data Multiplexing . . . . .	35
5.3 Rate Buffering and Data Priority. . . . .	35
5.4 General Information . . . . .	36
6. COMMAND LINK . . . . .	38
6.1 Command Transmitter . . . . .	38
6.2 Command Receiver. . . . .	39
6.3 Portable Command Box. . . . .	39
7. PRIME INTERFACE BOARD AND PRIME SOFTWARE . . . . .	42
7.1 Interrupt Handling. . . . .	43
7.2 VL DATA, VL STAT, IT STAT and Scratch Buffers. . . . .	43
7.3 Debugging Aids for the GPIB Interface and Telemetry Systems . . . . .	51

	page
8. LABORATORY REFERENCE MATERIAL. . . . .	57
9. REFERENCES . . . . .	59

## LIST OF FIGURES

	page
Figure 2.0.1 System Block Diagram. . . . .	5
Figure 3.0.1 Location of Microprocessor Card Cage. . . . .	8
Figure 3.0.2 Card Cage Ribbon Cable Assembly . . . . .	9
Figure 3.2.1 Format of Laser Mast Commands . . . . .	15
Figure 3.3.1 Switch Locations for Vehicle Control Parameters . . .	18
Figure 3.4.1 Microprocessor Display of Vehicle Parameters. . . .	21
Figure 4.0.1 Azimuth and Elevation Angles. . . . .	25
Figure 4.1.1 Mast Electronics Card Cage. . . . .	26
Figure 4.1.2 Laser Mirror. . . . .	27
Figure 4.3.1 Laser Detector and its Electronics. . . . .	30
Figure 5.0.1 Analog Multiplexer Board. . . . .	33
Figure 5.0.2 Telemetry Transmitter Board . . . . .	34
Figure 6.3.1 Portable Command Box. . . . .	41
Figure 7.2.1 Format of VL DATA Array. . . . .	45
Figure 7.2.2 Format of VL STAT Array. . . . .	47

## LIST OF TABLES

	page
Table 3.0.1 Configuration of the Microprocessor System. . . . .	10
Table 3.2.1 Valid Commands Decoded by the Microprocessor. . . . .	13
Table 3.3.1 Formulas for Switch-Selectable Vehicle Parameters used by Subroutine GETDAT . . . . . . . . . . . . . . . . .	19
Table 7.2.1 Format of ITSTAT Array. . . . . . . . . . . . . . . .	50
Table 7.3.1 GPIE Diagnostic Programs. . . . . . . . . . . . . . . .	52
Table 7.3.2 Microprocessor Transmitter Programs . . . . . . . . .	56
Table 8.0.1 Laboratory Notebooks. . . . . . . . . . . . . . . .	58

## LIST OF ACRONYMS

ADLC	<u>A</u> dvanced <u>D</u> ata <u>L</u> ink <u>C</u> ontroller
ACIA	<u>A</u> synchronous <u>C</u> ommunications <u>I</u> nterface <u>A</u> dapter
CSA	<u>C</u> enter of <u>S</u> canning, <u>A</u> zimuth Angle
DMA	<u>D</u> irect <u>M</u> emory <u>A</u> ccess
EOA	<u>E</u> nd- <u>O</u> f- <u>A</u> zimuth Interrupt
EOV	<u>E</u> nd- <u>O</u> f- <u>V</u> ehicle Interrupt
EOS	<u>E</u> nd- <u>O</u> f- <u>S</u> canning Interrupt
EPROM	<u>E</u> rasable <u>P</u> rogrammable <u>R</u> ead- <u>O</u> nly <u>M</u> emory
FIFO	<u>F</u> irst <u>I</u> n- <u>F</u> irst <u>O</u> t Memory
FSK	<u>F</u> requency <u>S</u> hift <u>K</u> eying
GPIB	<u>G</u> eneral <u>Purpose <u>I</u>nterface <u>B</u>oard</u>
IOS	<u>I</u> nput/ <u>O</u> utput <u>S</u> oftware
IPL	<u>I</u> mage <u>Processing <u>Laboratory</u></u>
JEC	<u>J</u> onsson <u>E</u> ngineering <u>C</u> enter
LSB	<u>L</u> east <u>S</u> ignificant <u>B</u> it
ML/MD	<u>M</u> ulti- <u>L</u> aser/ <u>M</u> ulti- <u>D</u> etector Scanning System
MSB	<u>M</u> ost <u>S</u> ignificant <u>B</u> it
NOP	<u>N</u> o <u>O</u> peration Instruction
PIA	<u>P</u> eripheral <u>I</u> nterface <u>A</u> dapter
PLL	<u>P</u> hase- <u>L</u> ocked <u>L</u> oop
PROM	<u>P</u> rogrammable <u>R</u> ead- <u>O</u> nly <u>M</u> emory
RAM	<u>R</u> andom <u>A</u> ccess <u>M</u> emory
RF	<u>R</u> adio <u>F</u> requency
SPU	<u>S</u> peed <u>U</u> nits

STU	<u>Steering Units</u>
TTL	<u>Transistor-Transistor Logic</u>
UAR/T	<u>Universal Asynchronous Receiver/Transmitter</u>
UFD	<u>User File Directory</u>

## **ABSTRACT**

This report is an overview of the complete electronics package which controls the Mars roving vehicle. It is meant to provide a broad overview of the systems which are part of that package and discusses some software debugging tools. The specific functions of the different electronic subsystems are described, with the microprocessor-based systems discussed more fully because they are not discussed in other reports in their present form.

PART 1  
INTRODUCTION

The Mars rover was developed at Rensselaer for the purpose of investigating the problems associated with an autonomous roving vehicle. If it were necessary to manually control a rover on Mars from Earth, the radio transmission time between the two planets would significantly limit its performance. Therefore it is essential that whatever control systems and computer programs used as a means of controlling the vehicle have the capability of operating without human intervention for extended periods of time. To this end, our rover has incorporated control elements through the use of a computer which is at a fixed location as well as electronic systems on the vehicle itself with the intent that on a real Mars exploration mission the fixed computer would either be on the Martian surface or else orbiting the planet so that transmission time could be neglected.

As of 1975, the fixed computer used was a Varian 620i located in the basement of the Sage Laboratory building. It was used to run the control algorithms which analyzed data from the rover. On board the 1975 rover was a predominantly analog control electronics package which provided feedback control of the vehicle's wheel speeds and formed the telemetry data stream sent to the fixed computer. This system had some inherent problems, however. First, the wheels often fought with each other when they were driving the vehicle since no provision was made for detecting if one or more of them might actually be dragging. Second, since the electronics was mostly analog cir-

cuitry, the drift which occurred between calibrations was a problem, as was the rather complex calibration procedure itself. Third, new ideas concerning control systems couldn't be experimented with because the basic system was realized totally in hardware.

It was decided that the control system should be redesigned in an attempt to alleviate these problems. The fixed computer which is presently used is a Prime 750 located on the second floor of the Jonsson Engineering Center (JEC) in the Image Processing Laboratory (IPL). The 750 was chosen for the new system because it has more speed than the Varian and the programs which run on it could be written in Fortran whereas before they were written in Varian assembler language. The use of Fortran has made it possible to develop a large part of the software on the IBM 3033 and subsequently transfer it to the Prime. Also, new software people joining the project are more likely to understand Fortran than assembler, thus simplifying the task of understanding the present Prime software.

The on-board control electronics package has been replaced by a completely digital system which is controlled by a Motorola M6800 microprocessor. The micro has the capability to monitor all wheel speeds independently and correct for both speed and torque variations on them. The feedback control for this purpose is digital in nature under the present system, so that once it has been calibrated, it should not be necessary to repeat the calibration procedures for some time, possibly never. Changes in the methods employed in the digital control system can be made easily due to the fact that such

changes would most probably be made in the program running in the microprocessor.

The laser hazard detection system installed on the 1975 rover was a single-laser/single detector system which had the problem of missing a significant number of the reflections from terrain in front of the vehicle. The new laser system is a multi-laser/multi-detector system which has the capability of firing up to 32 times the number of laser shots per scan as the old system as well as having better chances of receiving returns because of the fact that it has 20 detectors instead of only one, as did the 1975 version.

This report deals with the present control systems used on the rover as well as any related software tools and control programs which are applicable in this regard.

PART 2  
SIMPLIFIED OVERVIEW

A block diagram of the systems for controlling the rover and gathering data from it is shown in Fig. 2.0.1. Key elements are the laser mast, telemetry system, microprocessor, command link and the Prime computer. The laser mast gathers data concerning objects in front of the vehicle. This data is transmitted to the Prime by the telemetry system along with vehicle data so that the programs running on that machine can determine what course of action to take, for example, stopping the vehicle if it encounters a ravine or other hazardous terrain feature. The microprocessor must interpret commands sent from the Prime and also control the vehicle's wheel speeds. It can also display data about the state of the vehicle on a terminal attached to a port on the microprocessor itself. The possibility exists of sending commands to the laser mast from the Prime by having the microprocessor relay those commands, for example, it is possible to select the scan patterns used by the mast by sending the appropriate mast command. Should manual control of the vehicle be desired, a portable command box can replace the Prime computer as the source of commands. The General Purpose Interface Board (GPIB) is installed in the mainframe of the Prime computer and provides the circuitry for getting data into the Prime's memory where programs can use it as well as for sending commands to the vehicle. Through the use of radio frequency (RF) links, it is possible for the vehicle to operate approximately  $\frac{1}{4}$  mile from the fixed computer in

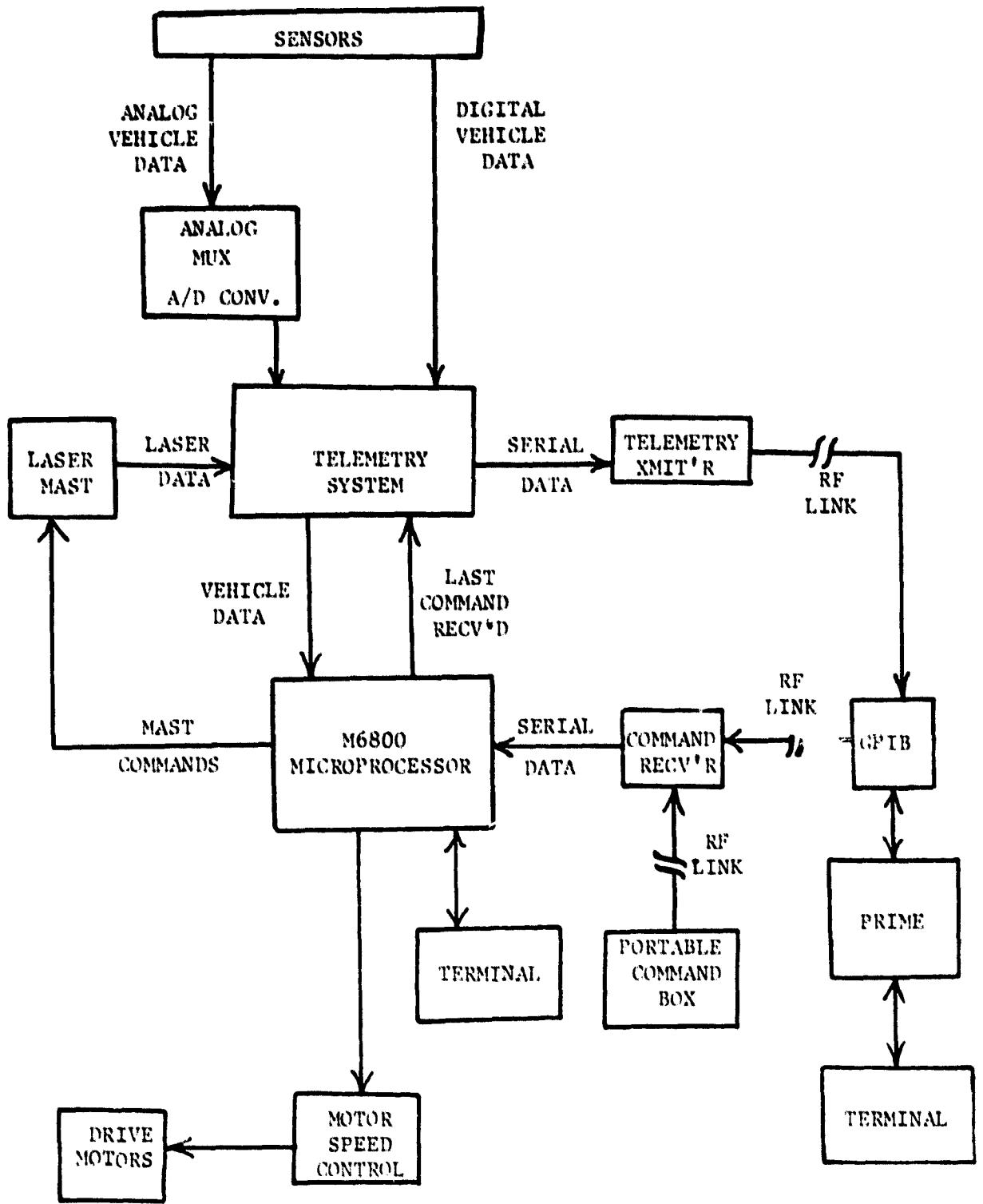


Fig. 2.0.1 System Block Diagram.

the Engineering Center.

Because of the fact that operation of the control system as a whole relies upon complex software and hardware interactions, testing the system provides numerous opportunities for exercising the GPIB. To that end, it has proven useful to have several programs which do nothing more than display data which the telemetry system is transmitting, even though this data is presented in its raw form. Still others exist which simulate commands which the path selection and navigation routines might send to the vehicle in an attempt to make sure that each command given to the microprocessor has the desired effect.

PART 3

MICROPROCESSOR CONTROL OF THE ROVER

The microprocessor on board the rover is housed in a card cage mounted in the rear of the vehicle's payload section. Its location is shown in Fig. 3.0.1. This card cage contains all of the electronics with which the microprocessor interfaces except the laser mast electronics, which are contained in a separate card cage mounted on the mast itself. Multi-conductor ribbon cables connect the microprocessor to these other electronic subsystems. Fig. 3.0.2 shows the card cage ribbon cable assembly.

The microprocessor functions to interpret commands from the Prime computer or the portable command box, maintain proper wheel speeds to result in a given vehicle speed and steering angle, display vehicle data on a terminal attached to one of its serial ports, format telemetry data for its own use, and interpret information concerning the torque on each wheel. Details of the microprocessor hardware and software are given in References 1 and 8.

When all of the circuit boards are installed in the rear vehicle card cage, the microprocessor is configured as shown in Table 3.0.1. Note that the locations from \$D400 through \$DFFF can be selected as corresponding to either Erasable Programmable Read-Only Memory (EPROM) or Random Access Memory (RAM). The dollar sign in this instance is used to indicate a hexadecimal address within the microprocessor system. All EPROM sockets are located on the microprocessor board except those for the A30 and A31 EPROMs, which are

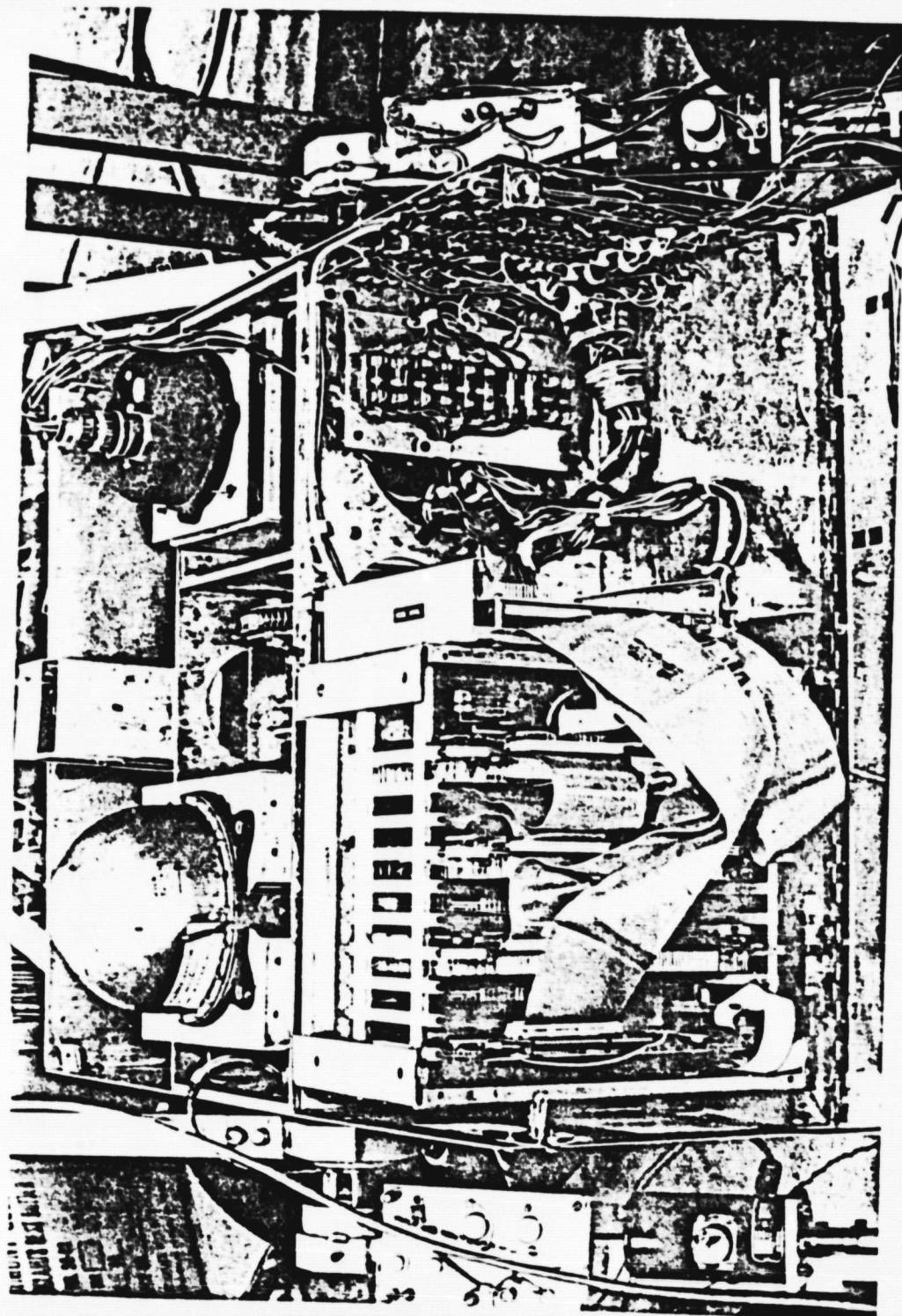


Fig. 3.0.1 Location of Microprocessor Card Cage.

Original page 5

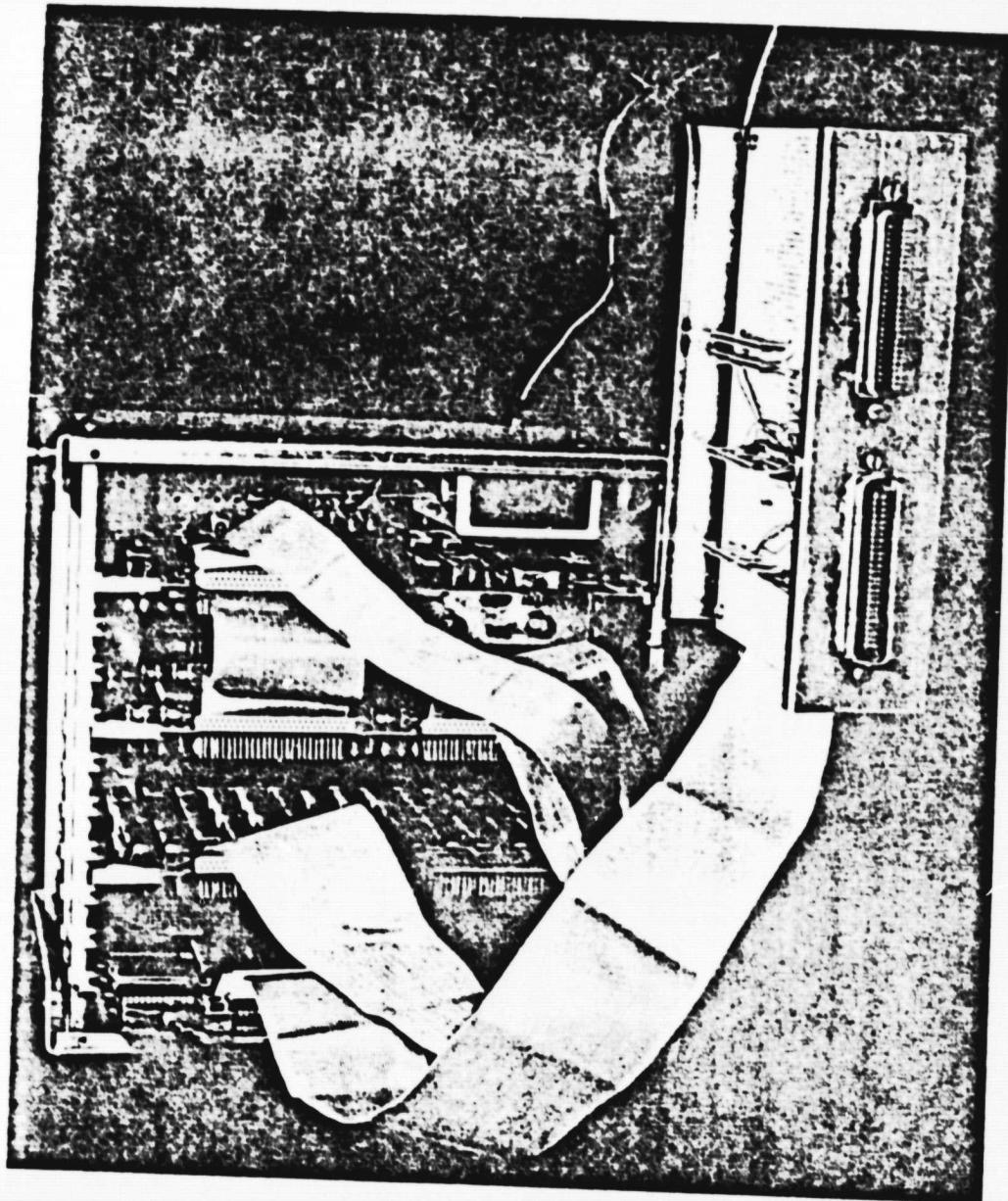


Fig. 3.0.2 Microprocessor Card Cage Ribbon Cable Assembly.

**Table 3.0.1 Configuration of the Microprocessor System.****A. Input/Output Ports****2 Peripheral Interface Adapters (PIAs)**

**PIA #1 (loc. \$8004)** - interfaces to EPROM programmer and telemetry system.

**PIA #2 (loc. \$8020)** - interfaces to EPROM programmer, laser mast and telemetry system.

**2 Asynchronous Communications Interface Adapters (ACIAs)**

**ACIA #1 (loc. \$8008)** - interfaces to terminal attached to the microprocessor.

**ACIA #2 (loc. \$8010)** - interfaces to the command link receiver.

**B. Ram Space**

Propulsion system scratchpad memory (loc. \$00 - \$FF)

MINIBUG\* monitor scratchpad memory (loc. \$A000 - \$A07F)

Additional RAM on RAM board (loc. \$B000 - \$BFFF)

(loc. \$D000 - \$DFFF)

Ram shared with the telemetry system (loc. \$C400 - \$C4FF)

**C. EPROM Space**

**2 Free EPROM sockets** (loc. \$D400 - \$D7FF)  
 (loc. \$D800 - \$DBFF)

**Input/Output Software (IOS) EPROM**  
 (supplied by Motorola with the micro)  
 (loc. \$DC00 - \$DFFF)

**MINIBUG EPROM**  
 (supplied by Motorola with the micro)  
 (loc. \$E000 - \$E3FF)

**"A30" monitor EPROM**  
 (developed in the rover lab to burn EPROMs)  
 (loc. \$9000 - \$93FF)

**A31" text EPROM**  
 (contains the text for messages printed out by the A30 monitor)  
 (loc. \$9400 - \$97FF)

\*The MINIBUG is a Motorola monitor program which starts running on the microprocessor when it is powered up.

located on the RAM board. Since the software for the propulsion system requires approximately 3K bytes (3072 locations) of EPROM space (three type 2708 EPROMs), the Input/Output Software (IOS) EPROM should be removed and the sockets corresponding to the continuous memory locations from \$D400 through \$DFFF should be used for the EPROMs containing that software. It should be noted that the Peripheral Interface Adapters (PIAs) do not interface to more than one device at a time, but may be connected to each of the devices they interface with via ribbon cables.

### 3.1 Program Organization

The program which runs in the M6800 was written to a large extent a year ago and is discussed at length in the report by J. Turner (Reference 1). The current version of the software is stored in the User File Directory (UFD) named <USERS3>MARS>GRAHAM>WHEELS.GD on the IPL's Prime 750 and should be on the backup tape which contains all the files used by the rover project. Also included in that UFD is a copy of the Motorola assembler and the emulator for the M6800 as well as the program \*DNLOAD which allows users to download programs from the Prime to the microprocessor. There are also several command files which are useful for assembling and reconstructing the file TAPE and LISTING. The former contains the M6800 machine code and the latter contains the listing of all of the subroutines which are part of the propulsion system software.

Another UFD named <USERS3>MARS>GRAHAM>SIM68 is used to store the copies of the M6800 assembler and emulator created when they

were being developed to run on the Prime instead of the IBM 3033. The assembler was furnished by Prof. J. McDonald, while the emulator has been modified from the standard Motorola version by including the capability to insert breakpoints in programs being emulated. There is also a copy of this modified emulator which ran on the IBM 3033. Note that SIM68 contains the source listing of the above programs, except the assembler, as well as some of the compiled code.

The propulsion system software itself consists of several subroutines which are stored in EPROMs and implement the functions described in the introduction to this chapter. A control loop whose function is to call the subroutines is copied from one of the EPROMs to the system's RAM during system initialization. Since the subroutine calls are in RAM, it is possible to bypass any of these subroutines by using a terminal attached to the micro to replace that subroutine call with "no operation" (NOP) instructions. After the propulsion system software has initialized itself, then typing any characters on that terminal will return the user to the MINIBUG monitor on the microprocessor, thus enabling him to change any memory locations or load additional programs before executing that software again. Only a single character must be typed to break the control loop.

### 3.2 Command Decoding

The list of valid commands which the microprocessor can interpret is shown in Table 3.2.1. These are decoded in a subroutine named CMDDEC, which has replaced Turner's NEWCMD subroutine. In this new routine, it is possible to decode both single-byte eight-bit and

Table 3.2.1 Valid Commands Decoded by the Microprocessor.

Command Name		Command Code (Hex)
ONE WHEEL DRIVE	Off	08
	Left Rear	0C
	Right Rear	0D
	Left Front	0E
	Right Front	0F
STEERING: (LEFT)	-90.0°	47
	-67.5°	46
	-56.25°	45
	-45.0°	44
	-33.75°	43
	-22.5°	42
	-11.25°	41
	0.0°	40
(RIGHT)	0.0°	48
	11.25°	49
	22.5°	4A
	33.75°	4B
	45.0°	4C
	56.25°	4D
	67.5°	4E
	90.0	4F
MAIN DRIVE	Forward Full	53
	Forward Two-thirds	52
	Forward One-third	51
	Stop	50
	Stop	54
	Reverse One third	55
	Reverse Two-thirds	56
	Reverse Full	57
FRONT WHEEL DRIVE :Forward		7A
	Stop	78
	Reverse	7B
MISC. COMMANDS:	Gyro Init	77
	Vehicle Reset	7F
	Display On	76
	Display Off	74
	Override Terrain Compensation	10
	Restore Terrain Compensation	11-1F
TWO-BYTE STEERING		8X nn
	where nn is the two's-complement steering angle in steering units (STU).	
TWO-BYTE MAST COMMAND		9n nn
	where nnn are bits sent to the laser mast.	

two-byte eight-bit commands, whereas on the 1975 rover only seven-bit single-byte commands could be decoded. One reason for changing to eight-bit commands was that the most significant bit (MSB) now indicates whether the command is one or two bytes long. A zero in the MSB signifies a single-byte command, while a one in the MSB signifies a two-byte command and the microprocessor interprets the next command byte which it receives as the second half of that command. This format keeps the commands used with the 1975 rover compatible with those used with the 1980 vehicle. This was desirable so that time-consuming hardware modifications would not have to be made to the portable command box used for manual control of the vehicle.

The first type of two-byte command is the two-byte steering command. This exists so that it is possible to choose any steering angle from -90 to +90 degrees and not be limited by the fixed steering angles provided by the single-byte steering commands. The first byte of this command contains the hexadecimal characters 8X, where X indicates four bits which are "don't-cares." The second byte contains the desired steering angle in steering units in two's-complement form. To convert from degrees to steering units, multiply by 1.421875. This conversion factor results from representing an angle between -90 and +90 degrees as a two's complement binary number between -127 and +127. The Prime computer generates two-byte steering commands but the portable command box still sends single-byte steering commands.

The second form of two-byte command which is also sent exclusively by the Prime is the laser mast command, the format of which is shown in Fig. 3.2.1. The actions which the mast takes when it

## FIRST BYTE



- A. Center of Scan Azimuth Command**  
Mast Command Bits = 0000

Second byte of the command contains the azimuth angle, 0 through 255, of the center-of-scan.

- B. Scan Pattern/Laser Enable Command**  
Mast Command Bits = 0001

## SECOND BYTE



1 - laser enabled  
0 - laser disabled

00 - first elevation pattern  
01 - second elevation pattern  
10 - third elevation pattern  
11 - fourth elevation pattern

00 - first azimuth pattern  
01 - second azimuth pattern  
10 - third azimuth pattern  
11 - fourth azimuth pattern

- C. Scanning Speed Command**  
Mast Command Bits = 0010

## SECOND BYTE



00 - 0 scans/sec.  
01 - 0.25 scans/sec.  
10 - 0.50 scans/sec.  
11 - 1.0 scans/sec.

Fig. 3.2.1 Format of Laser Mast Commands.

receives one of these commands are described in Section 4.1.

### 3.3 Wheel Speed Control

It is desirable to control the speed of each wheel individually from two points of view. The first is to take into account the desired steering angle of the vehicle and change the wheel speeds accordingly. Since the front axle of the 1980 rover no longer has a steering motor as the 1975 version did, it relies on differences in the front wheel speeds as the only means for steering the vehicle. The subroutine which corrects the wheel speeds for an appropriate steering angle is named STEERCOR. Its present form is essentially that described in Reference 1.

The second reason for controlling the wheel speeds is to attempt to keep them from working against one another and wasting battery power. A subroutine named TERCOR monitors switches on the drive train of each wheel which measures whether the torque on that wheel is positive or negative. If a wheel is found to be dragging (negative torque), then adjustments are made in that wheel's speed until it does not drag. This is done by increasing a parameter called the set speed for that wheel. Each time the microprocessor detects that the wheel is dragging, it will increase the set speed by 38 speed units (SPU), which corresponds to a velocity of 0.15 m/sec. in the forward direction.

Once the set speed parameters for each wheel are generated by STEERCOR and TERCOR, they are digitally low pass filtered by the subroutine FILTER. These filtered set speeds are used by the propo-

tional speed controller subroutine CONTROL to generate the drive signals which are fed to the motor driver circuitry for each wheel.

CONTROL calculates the difference between the filtered set speed and the actual wheel speed for each wheel, multiplies this by the appropriate controller gain, and adds or subtracts the resulting change in the drive signal to the present drive signal. It checks that the maximum values for negative and positive wheel speeds are not exceeded in the process of this calculation. The controller gains are switch-selectable parameters, the switches for which are located on the motor speed board in the microprocessor card cage.

Fig. 3.3.1 shows the location of these switches as well as those used to set the filter parameters of the digital filter. It is good practice to set the filter parameters and controller gains with the vehicle's wheels off the ground and the display routine described in Section 3.4 running in order to monitor the numerical values of those parameters.

Each of the switches in Fig. 3.3.1 is divided into two sections of four bits each. After reading them, the subroutine GETDAT calculates the controller gains, filter parameters and the three velocities at which the vehicle can travel as shown in Table 3.3.1. These three velocities are the full, two-thirds and one-third speeds, calculated in speed units (SPU). The circuitry which interfaces the microprocessor to the drive motors has been changed since Turner's original design and is discussed in Reference 8. Suggested switch settings are discussed in Reference 1.

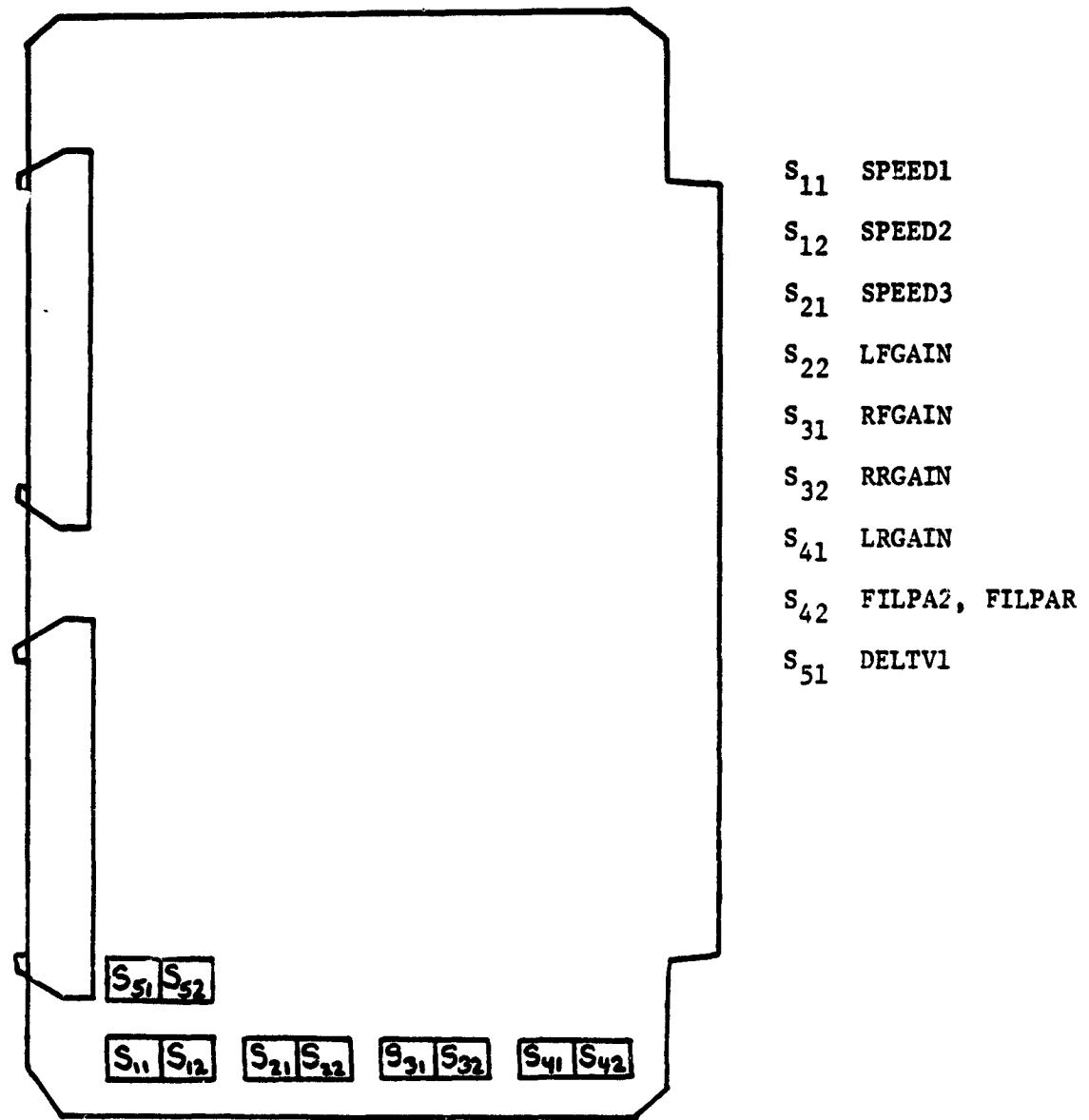


Fig. 3.3.1 Switch Location on Motor Speed Board for Vehicle Control Parameters.

**Table 3.3.1 Formulas for Switch-Selectable Vehicle Control Parameters used by Subroutine GETDAT.**

Parameter	Formula	Comments
Speed1	$25+4*DS_{11}$	'One-third' speed (SPU)
Speed2	$25+4*DS_{12}$	'Two-third' speed (SPU)
Speed3	$25+4*DS_{21}$	'Full' speed (SPU)
LFGAIN	$4*DS_{22}$	Left front controller gain
RFGAIN	$4*DS_{31}$	Right front controller gain
RRGAIN	$4*DS_{32}$	Right rear controller gain
LRGAIN	$4*DS_{41}$	Left rear controller gain
FILPA2	$0.875+DS_{42} * 0.00078125$	Filter time constant
FILPAR	$1-FILPA2$	Filter gain
DELTIV1	$4*DS_{51}$	Maximum turn rate

Note:  $DS_{ij}$  is the setting of the switch for the selected parameter as shown in Fig. 3.1.1

### 3.4 Display of Vehicle Parameters

The subroutine DISPLAY will display certain information about the state of the vehicle on a terminal connected to the microprocessor. The microprocessor responds to commands which can cause it to enable or disable this display. While any type of terminal may be used for this purpose, a CRT type running at 9600 baud should give the best performance as it provides a fairly fast update of the screen, the format of which is shown in Fig. 3.4.1. The amount of data displayed by this subroutine has been increased since Turner's original version to display almost all of the dynamic variables used by the microprocessor. In the figure, numerical data which is printed out is shown as 'nnn.' The steering angle is printed in degrees and the wheel speeds are printed in mm/sec as the linear speed of the center of each wheel.

### 3.5 Vehicle Data Acquisition

The microprocessor relies on the telemetry system to gather the vehicle data it needs. The vehicle data is fed through an analog multiplexer to an analog-to-digital converter. The telemetry system makes this digitized analog data, along with any digital vehicle data, available to the microprocessor by storing the data in a Random Access Memory (RAM) which is shared by the telemetry system and the microprocessor.

Currently the microprocessor has control over who has read and write access to that RAM; thus it has the responsibility of allowing the telemetry system to write to it. The software in use

<b>DESIRED STEER:</b>	$\frac{+nnn}{-nnn}$	<b>DEGREES</b>			
<b>ACTUAL STEER:</b>	$\frac{+nnn}{-nnn}$	<b>DEGREES</b>			
<b>DESIRED SPEED:</b>	$\frac{+nnn}{-nnn}$	<b>MM/SEC</b>			
<b>WHEEL SPEEDS:</b>	LF $\frac{+nnn}{-nnn}$	RF $\frac{+nnn}{-nnn}$	RR $\frac{+nnn}{-nnn}$	LR $\frac{+nnn}{-nnn}$	
<b>DESIRED</b>	$\frac{+nnn}{-nnn}$	$\frac{+nnn}{-nnn}$	$\frac{+nnn}{-nnn}$	$\frac{+nnn}{-nnn}$	
<b>ACTUAL</b>	$\frac{+nnn}{N}$	$\frac{+nnn}{N}$	$\frac{+nnn}{N}$	$\frac{+nnn}{N}$	
<b>DRAGGING</b>					
<b>WORM GEAR SWITCHES:</b>	nn	<b>DRAG INDICATOR WORD: nn</b>			
<b>GYRO POT ANGLE:</b>	nnnn	<b>GYRO STATUS:</b>	nn	<b>GYRO TIMER:</b>	nn
<b>SET SPEEDS - ONE:</b>	nn	<b>TWO:</b>	nn	<b>THREE:</b>	nn
<b>CONTROLLER GAINS - LF:</b>	nn	<b>RF:</b>	nn	<b>RR:</b>	nn
<b>FILTER GAIN:</b>	nn	<b>TIME CONSTANT:</b>	nn		
<b>MAXIMUM TURN RATE:</b>	nn				
<b>COMMAND WORD:</b>	nnnn				

Fig. 3.4.1 Microprocessor Display of Vehicle Parameters.

at the present time grants this access to the telemetry system for the majority of the time that it is running. The microprocessor will rescind that access only while it is reading the RAM and writing to it for its own purposes, a process which happens only once each time through the control loop and which takes 100-120  $\mu$ sec. every 10 msec. Thus the telemetry system is denied access to that RAM for approximately 1% of the time when both the microprocessor and telemetry system are running.

## PART 4

### LASER MAST

The laser mast provides the means by which the Prime computer can detect objects in the vehicle's path. The mast supports optics and electronics which allow its laser to fire a matrix of laser shots on the terrain in front of the vehicle. While it is possible to scan the terrain behind the rover, this is not practical due to the range of the laser scanning system and the amount the vehicle extends behind the mast. Documentation on the mast control electronics may be found in References 3 and 6. Documentation on the laser and detector electronics can be found in Reference 7.

Laser shots can be fired so as to form a straight line radiating outward from the mast on level ground. Each such line of laser shots is referred to as an azimuth because it occurs at a particular azimuth angle with respect to a zero-degree reference directly ahead of the vehicle. Since there are 256 possible azimuth angles and the mast rotates through 360 degrees for each scan, it is possible to fire an azimuth of laser shots anywhere with a resolution of 1.4 degrees. A maximum of 32 of the available 256 azimuths may be used in any particular scan.

Within each azimuth it is possible to have up to 32 laser shots which are known as elevations because they are laser shots which are fired at different angles in a vertical plane which contains the mast. Each elevation will correspond to a spot on the terrain about the vehicle at a certain radial distance from the mast along that

azimuth angle. The resolution of elevation angles is 0.35 degrees. Azimuth and elevation angles are depicted in Fig. 4.0.1.

#### 4.1 Electronic Mast Controller Capabilities

The multi-laser/multi-detector (ML/MD) laser scanning system is controlled by an electronics package located in the mast electronics card cage mounted on the back of the laser mast and shown in Fig. 4.1.1. It was originally designed by Craig (Ref. 3) during the academic year 1977/78. Its present form has not changed significantly from the original design, except as noted in Reference 6.

Fig. 4.1.2 shows the rotating mirror located at the top of the mast. This mirror is used to generate different elevation angles within a particular azimuth of laser shots. The speed of the mirror is controlled by phase-locked loop (PLL) circuitry such that it spins at a rate which is 24 times that of the laser mast as a whole. A shaft encoder is connected to the mirror and provides a pulse output as feedback to the PLL. The same type of PLL and shaft encoder arrangement controls the speed of rotation of the entire mast.

By sending laser mast commands from the Prime computer via the command link to the microprocessor on board the vehicle it is possible to change the scanning speeds of the laser mast. Available speeds are 0., 0.25, 0.5 and 1.0 scans per second. The ratio of mast velocity to that of the mirror is always 1 to 24. It is also possible to turn off the mirror or mast drive motors independently.

In addition to controlling the mast and mirror velocities, the mast controller also controls the pattern of the laser shots for

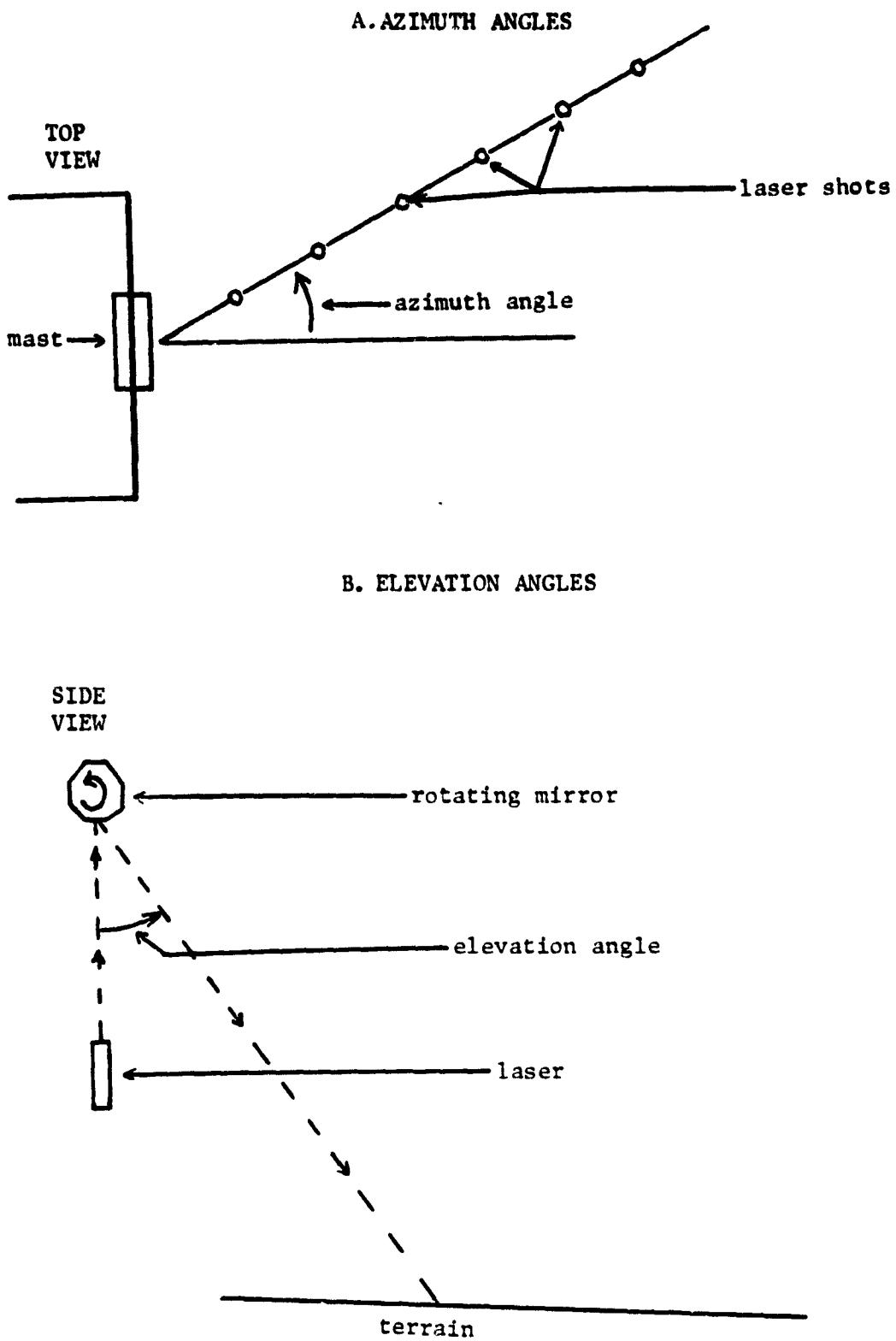


Fig. 4.0.1 Azimuth and Elevation Angles.

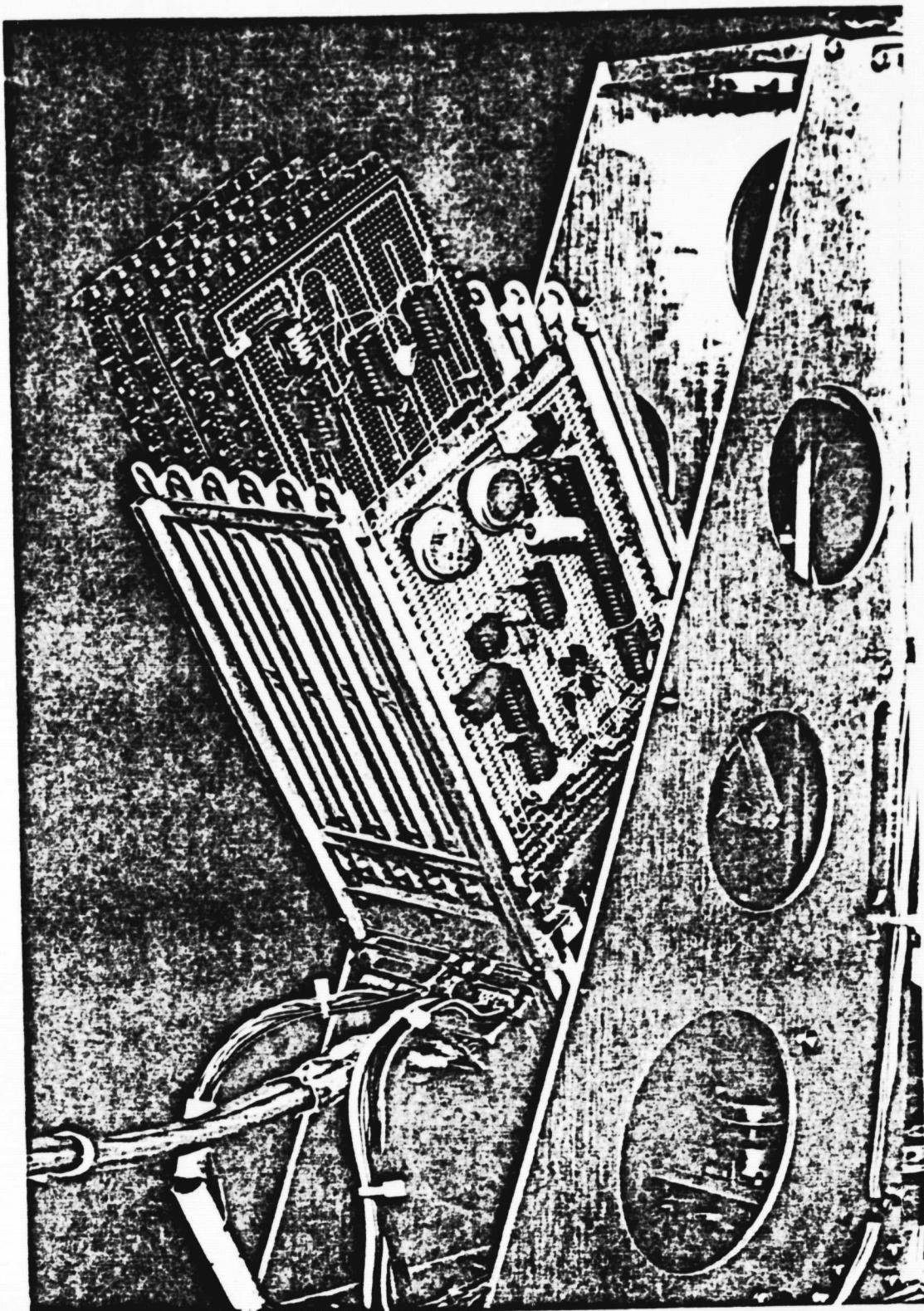


Fig. 4.1.1 Mast Electronics Card Cage.

ORIGINAL PAGE IS  
OF POOR QUALITY

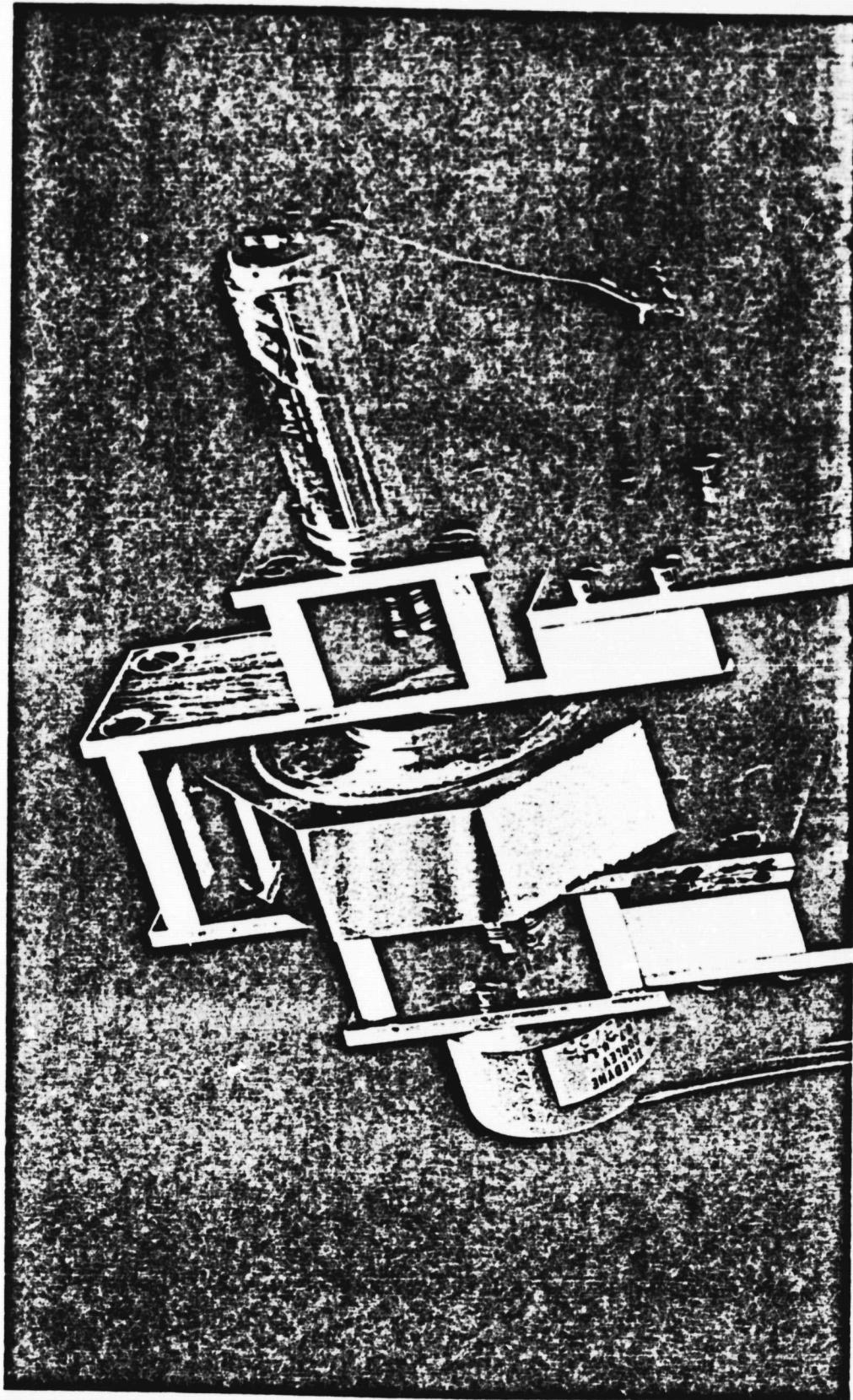


Fig. 4.1.2 Laser Mirror

each scan. Both the elevation and azimuth angles at which laser shots occur are determined by data stored in two type 2708 EPROMs, one for elevation and one for azimuth angles. The system is set up so that there are 256 possible elevation angles and 256 possible azimuth angles. Since each type 2708 EPROM has 1024 locations, the elevation and azimuth EPROMs can store up to four different elevation and azimuth patterns each. The elevation and azimuth patterns being used can be selected by switches on the memory board in the mast electronics card cage.

The capability exists to offset the entire set of azimuth angles by a fixed angle called the center-of-scan azimuth (CSA) angle. The CSA angle can be any of the possible azimuth angles, and can only be changed by a command received from the Prime. It is also possible to specify a fixed offset angle for elevation angles by setting switches on the elevation board.

#### 4.2 Laser Mast Commands

The ML/MD controller electronics has been designed to accept several commands from the Prime computer via the command link. This would enable a program running on the Prime to change the dynamic operation of the laser mast to suit conditions which the rover might encounter. At the present time, there are three commands to which the mast electronics will respond. These are the command to change the CSA angle, the command to select scan patterns and enable the laser, and the command to specify the scanning speed of the mast. They are explained further in Fig. 3.2.1 and Reference 6.

#### 4.3 Laser Electronics and Power Supply

The laser on the mast is a pulsed type and as such can generate considerable noise spikes in other electronic systems, particularly in the detector electronics. The power supply for the laser has been located in close proximity to the laser itself to minimize interference generated by longer cabling. The addition of bypass capacitors at key points, enabling the detector electronics only at the instants when the laser fires, and other measures described in Reference 7, have reduced the noise to an acceptable level.

#### 4.4 Laser Data Description

Near the base of the rotating portion of the laser mast is the detector, shown in Fig. 4.3.1, which detects reflections of laser shots from terrain in front of the vehicle. In the figure is shown a 135-mm camera lens behind which is mounted a 20-element photo-diode array. Shown above the lens-photo-diode assembly is the detector electronics, discussed in Reference 7.

Briefly, if a reflection occurs from the surrounding terrain within the field of view of the detector, which is about 30 degrees, then it will fall on one or more of the photo-diodes in the array. The detector electronics looks for pulses at the outputs of the diode array only when the laser fires and feeds its output to a priority network which determines the identification numbers of up to two of the diodes which received the reflection. Most returns will fall on a maximum of two diodes. Due to the design of the

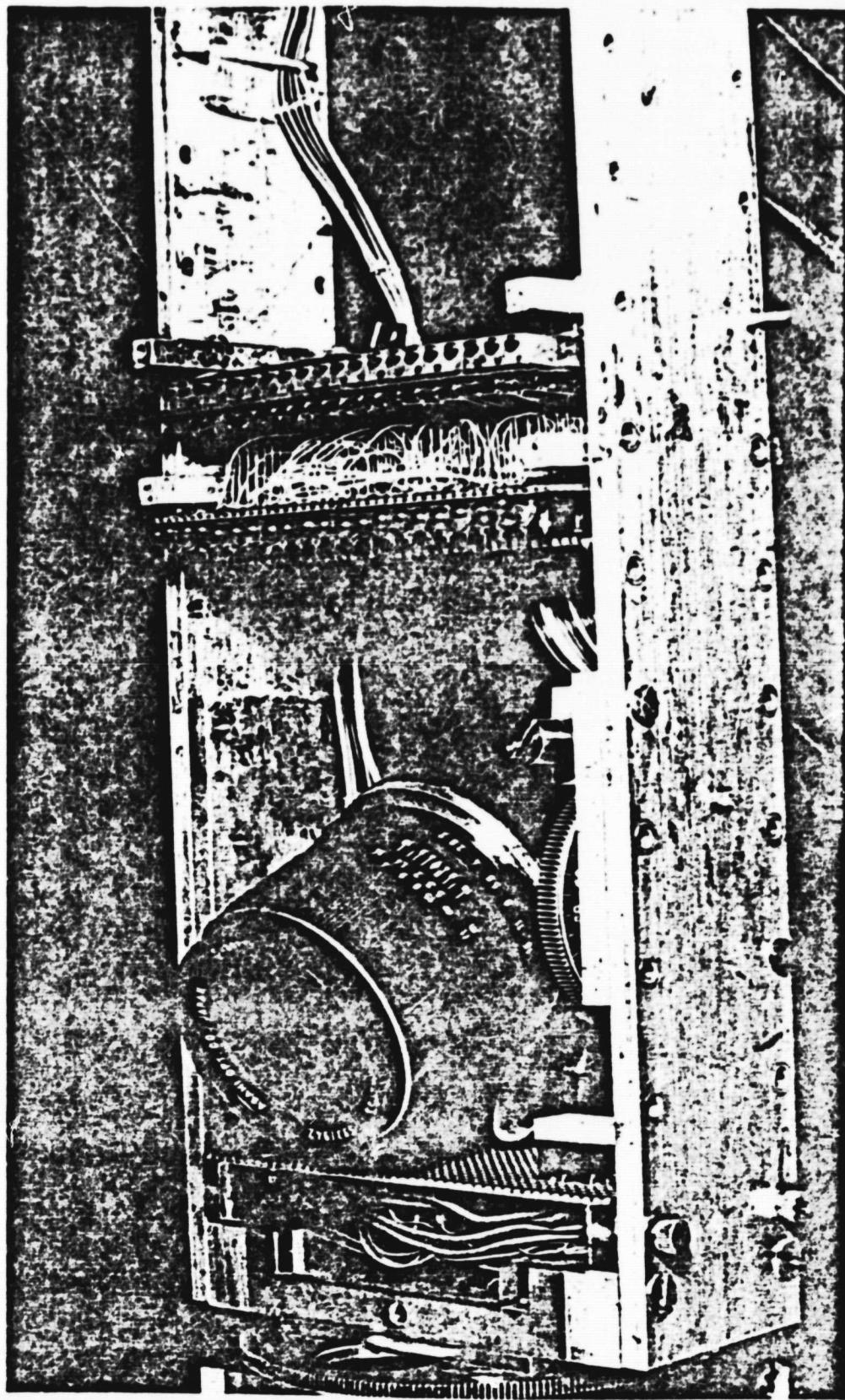


Fig. 4.3.1 Laser Detector and its Electronics.

ORIGINAL PAGE IS  
OF POOR QUALITY

priority network, when two or more detectors receive a return from a laser shot, the two identification numbers which are generated are those of the highest and lowest detector which received that return.

The laser data sent to the Prime computer has the form of two 16-bit words for each laser shot. First, a 16-bit address indicates which azimuth and which elevation the shot was fired at, and second, a 16-bit data word indicates which detector diode(s) received a return from that shot, if any. The 16-bit data word actually contains only 10 bits of valid information. The six most significant bits (MSBs) are all ones while the ten least significant bits (LSBs) contain two 5-bit words which are the identification numbers of the detector diode(s) receiving the return. Since there are only 20 diodes in the array, these identification numbers will be between 1 and 20 inclusive when a return is received. If no return is received, then the 5 LSBs of the data will be all zeros.

Data coming from the priority network is stored in First In-First-Out (FIFO) memories because the mast can generate data faster than the telemetry system can transmit it to the Prime computer for analysis. The FIFOs buffer the rates at which data is generated and transmitted. The telemetry system gives priority to the transmission of laser data over transmission of vehicle data such as wheel speeds or gyro angles.

## PART 5

### TELEMETRY SYSTEM

It is the function of the telemetry system to gather all analog and digital vehicle data as well as the digital data from the laser mast, make it available to the microprocessor, and transmit it to the Prime computer. There is provision for up to 16 analog and 16 digital channels of vehicle data, the sampling of which is controllable by a type 2708 EPROM on the analog multiplexer board shown in Fig. 5.0.1. The analog multiplexer board is connected by ribbon cable to the telemetry transmitter board, shown in Fig. 5.0.2, which formats the telemetry data into a serial stream for transmission to the Prime computer. It is the responsibility of the telemetry system to supply vehicle data not only to the Prime computer via the radio frequency (RF) telemetry link, but also to the microprocessor on board the vehicle via the RAM shared by both of them. The hardware is described by Cipolle (Reference 2).

#### 5.1 Data Format

The data sent to the Prime computer is made up of a 16-bit address word sent with each 16-bit data word. The address word tells the Prime where that data word came from on the vehicle or what laser shot the data word corresponds to if it came from the mast. Each address word contains three interrupt bits which cause the Prime to perform certain operations in regard to where in the user's address space the data is stored.

The three interrupts which the Prime recognizes are the

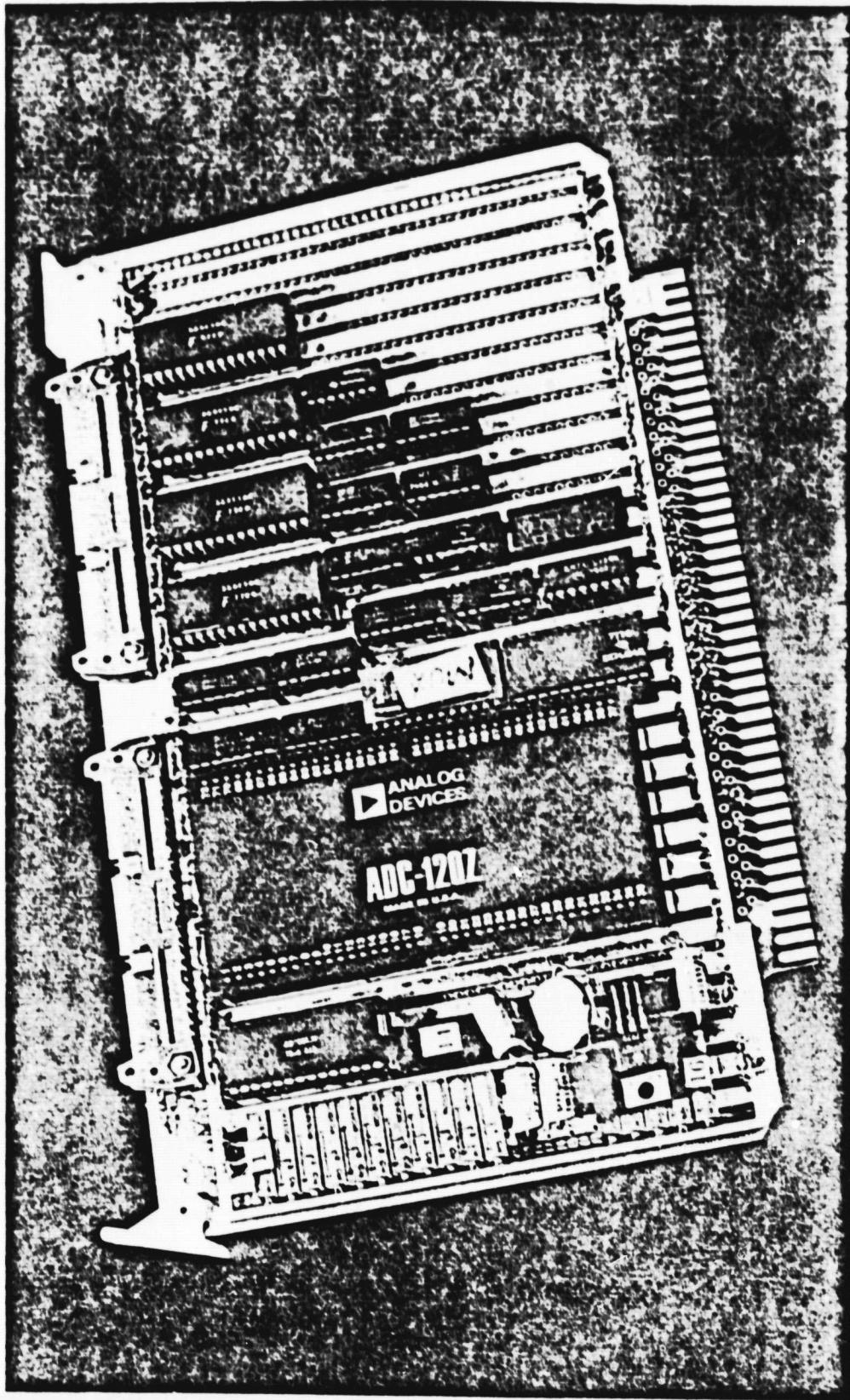


Fig. 5.0.1 Analog Multiplexer Board.

ORIGINAL PAGE IS  
OF POOR QUALITY

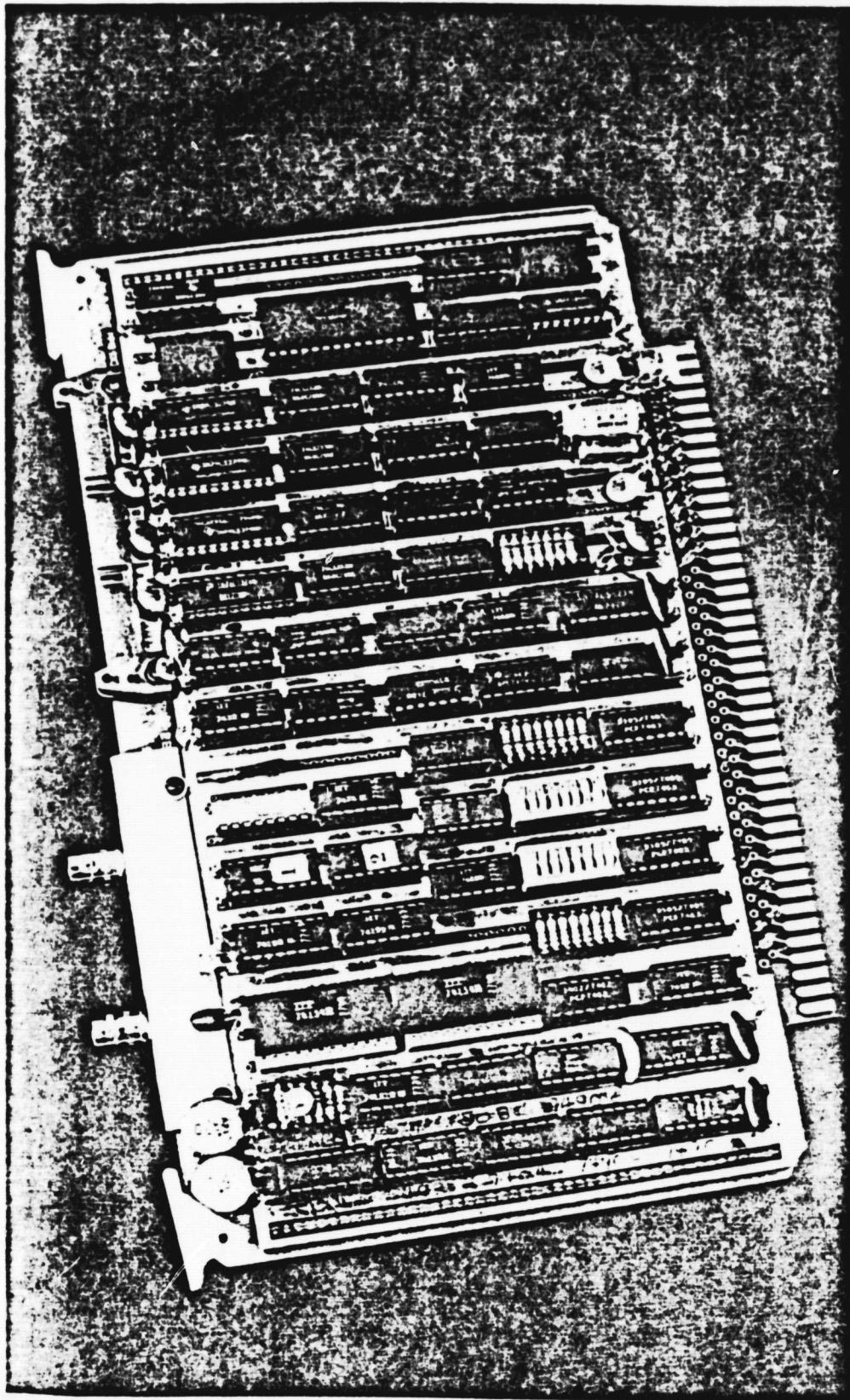


Fig. 5.0.2 Telemetry Transmitter Board.

ORIGINAL PAGE IS  
OF POOR QUALITY

end-of-azimuth (EOA), end-of-vehicle (EOV), and end-of-scan (EOS). An EOA interrupt is sent as part of the address for the last laser data word for each group of shots in an azimuth generated by the laser mast. An EOV interrupt is generated as part of the address for the last vehicle data word in a group of those words sent to the Prime. Finally, an EOS interrupt is generated by the laser mast as part of the address sent with the last laser data word of each scan made by the laser mast. For an explanation of the elevation/azimuth scanning concept used by the mast, see Part 4.

### 5.2 Vehicle and Laser Data Multiplexing

The telemetry system must multiplex the laser data coming from the mast with 32 vehicle data words to be sent back to the Prime computer. The software which drives the interface in the mainframe of the Prime expects 32 vehicle words to accompany, that is immediately precede, each section of 32 laser words for each azimuth. Each azimuth need not contain 32 elevations and it is not necessary to send the vehicle data with each azimuth, but doing the latter will ensure that valid vehicle data are always stored with any particular section of laser data. Although the vehicle data words sent back to the Prime are 16 bits long, none of the data sent occupies all 16 bits except the command word echo data.

### 5.3 Rate Buffering and Data Priority

The data from both the vehicle and the laser mast are fed into FIFO memories in order to allow the telemetry system to transmit data at a different rate from that at which they are collected. The

absence of the FIFOs to buffer these two rates would have forced the synchronous operation of all of the systems on the vehicle which would have hindered design tasks considerably and greatly reduced their flexibility as a research tool. Data coming from the laser mast must have priority over the vehicle data when both are present, ensuring that none of the data from a laser scan are lost.

#### 5.4 General Information

The final serial output of the telemetry transmitter is generated by a Motorola chip called an Advanced Data Link Controller (ADLC) from the parallel data presented to it. The format of the transmission along with the mode of the transmitter section of the ADLC is controlled by type 8223 Programmable Read-Only Memories (PROMs) which are documented in the report by Cipolle (Reference 2). Each frame transmitted by the ADLC contains 16 address bits, 16 data bits, and 32 bits relating to starting the frame, error checking, and frame termination. The error checking code provides the ADLC which receives the data at the other end of the telemetry link with the capability to perform cyclic redundancy error checking on each frame.

The telemetry receiver is constructed on a portion of the GPIB interface in the mainframe of the Prime computer. The output of the ADLC which receives the serial data is in parallel form and is used to fill FIFOs whose outputs are placed in the user's address space via direct memory access (DMA) transfer.

While it is intended that the link from the telemetry transmitter board to the Prime be an RF link, presently it is made by co-

axial cable. When the RF link is implemented, cables will still run from the Prime to the rover lab because the radio receiver for the telemetry data will be located in the rover lab with its antenna on the roof of the Engineering Center.

## PART 6

### COMMAND LINK

The command link is the means by which programs running on the Prime analyzing data from the mast and the vehicle may direct the vehicle's actions so that it may travel toward its target while avoiding obstacles in its path. The command link consists of a Universal Asynchronous Receiver/Transmitter (UAR/T) on the GPIB interface which generates a serial data stream that is sent upstairs to the rover lab over a multi-conductor cable. In order for this cable to attach to the GPIB, a short length of coaxial cable with a BNC connector was used in the Image Processing Lab (IPL) where the Prime is located. The radio frequency (RF) transmitter for the command link is in the rover lab and has an antenna on the roof of the Engineering Center. An RF receiver is located on the vehicle and its output is fed through demodulator circuitry to the microprocessor.

#### 6.1 Command Transmitter

On the GPIB interface, the UAR/T converts the parallel data commands from the Prime into serial form. The resulting transistor-transistor logic (TTL) level signal is buffered and sent via cable to the RF transmitter in the rover lab. The RF transmitter also incorporates a Frequency Shift Keying (FSK) modulation circuit which converts the TTL input to an audio signal before it is transmitted to the rover.

## 6.2 Command Receiver

The RF receiver on board the vehicle is mounted inside a small box with an antenna on it in a central position on the vehicle. This enclosure also contains an FSK demodulator which takes the received audio signal and converts it back into a TTL level serial data stream. The box operates on 12 VDC and contains the necessary voltage regulators to produce the correct voltages for the components inside. A speaker has been provided to monitor the quality of the audio signal being received by the RF receiver.

The receiver box has two outputs which are fed to the Asynchronous Communications Interface Adapter (ACIA) on the microprocessor board which receives commands. The first output is the serial data output which is a TTL level signal that the ACIA can accept as an input. The second output is the loss-of-signal output, which remains in the low state as long as a strong, clean signal is being received.

## 6.3 Portable Command Box

If it is desirable to test the rover under manual control, this can be done by using the portable command box to send commands to the vehicle. It contains the necessary radio transmitter and FSK modulation circuitry to communicate with the rover in the same manner as the Prime does. The box has been modified to send eight-bit commands with even parity and one stop bit since this is the format of the serial data stream used for commands to the 1980 rover. The most significant bit (MSB) is always a zero, since the box

cannot transmit two-byte commands. The portable command box is shown in Fig. 6.3.1.

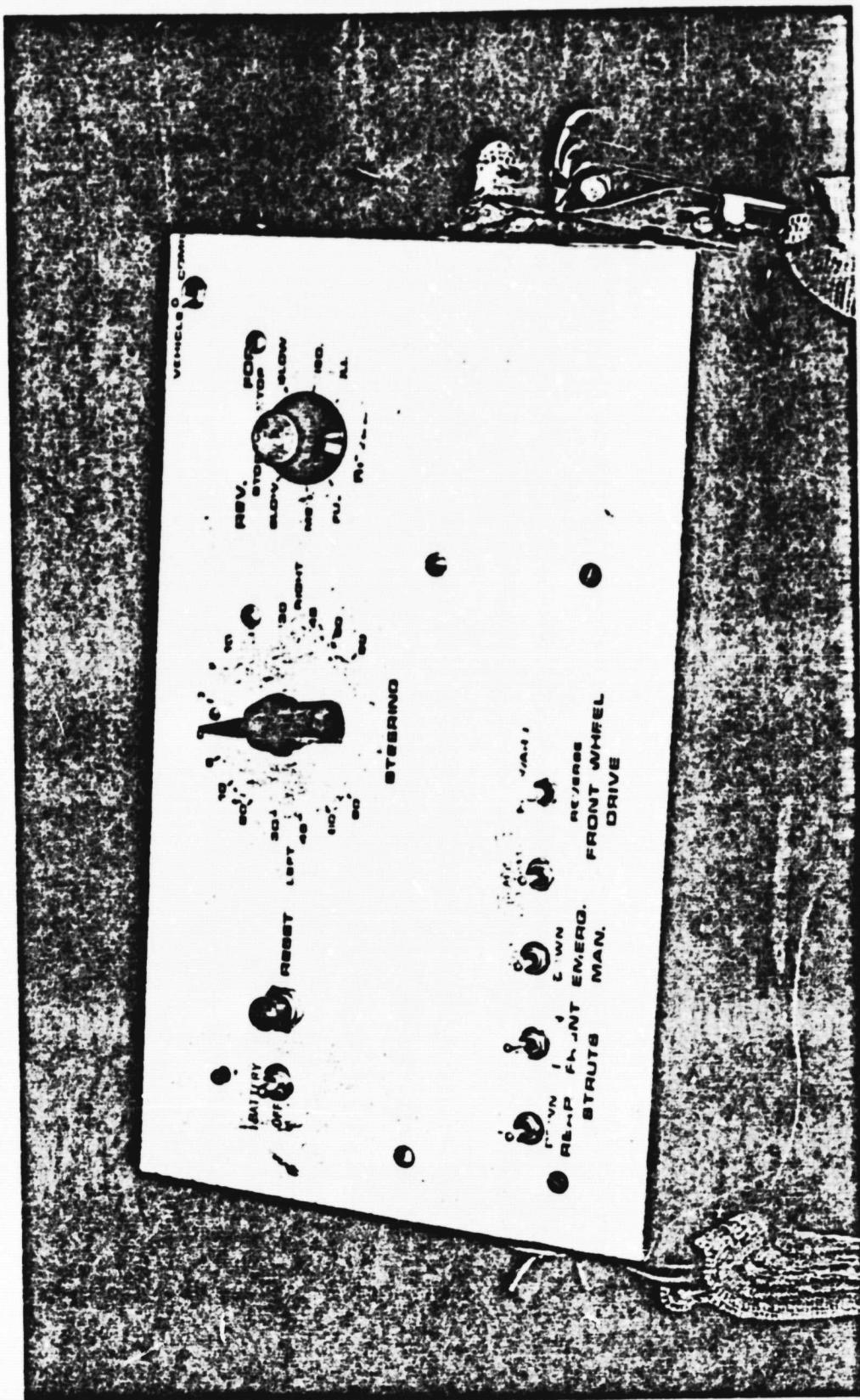


Fig. 6.3.1 Portable Command Box.

## PART 7

### PRIME INTERFACE BOARD AND PRIME SOFTWARE

The General Purpose Interface Board (GPIB) on the Prime provides a means for getting information into that computer at a high data rate via direct memory access (DMA) transfer. As mentioned in Section 5.1, each 16-bit data word is accompanied by a 16-bit address word which indicates where that data came from on the rover. Present in that address word are three interrupt bits which cause the Prime to perform certain operations as it is storing the rover data.

Due to the complex nature of the Prime's operating system, a user's program cannot simply manipulate the GPIB but must call a special PRIMOS\* system subroutine named TSROVR that was developed by M. Potmesil (Reference 5). This routine is an input/output (I/O) driver which handles all interactions with the interface board. By using TSROVR the user's access to the GPIB is easily accomplished.

The software that has been written for the Prime falls into three categories. First, the realtime software which processes the data from the rover and sends appropriate commands back to it; second, the diagnostic programs written to debug the telemetry system and the laser mast itself, and third, the assembler I/O driver TSROVR written by Potmesil. The realtime software is described in References 9 and 10. The diagnostic programs are mentioned in Section 7.3 and are completely documented in the notebook entitled

\*PRIMOS is the operating system used on the Prime computer.

"GPIB," volume 2, in the laboratory.

This chapter discusses some of the physical properties of the GPIB board, some of the more obscure details of the operation of T\$ROVR, and the functions of several programs which were written as diagnostic tools for debugging not only the GPIB itself but the entire system. A complete description of the hardware can be found in Reference 4.

### 7.1 Interrupt Handling

Upon the start of execution of a user's program on the Prime, a call must be made to T\$ROVR to initialize itself and the GPIB. After this initialization, T\$ROVR waits for an EOS interrupt before making any data available to the user. Therefore it is imperative that the telemetry system be transmitting these interrupts from time to time. Once an EOS has been received, T\$ROVR starts to fill the first buffer which the user has specified to hold the telemetry data. From that point on, every time an EOA interrupt is sensed, T\$ROVR will start to fill the next section of the buffer until the buffer is full. Subsequent EOS interrupts will declare the buffer as being full and T\$ROVR will start to fill the next buffer if it is empty. It is important to note that an EOS can occur at any time, and that the occurrence of that EOS is the only determining factor for declaring a buffer as being full.

### 7.2 VLDATA, VLSTAT, ITSTAT and Scratch Buffers

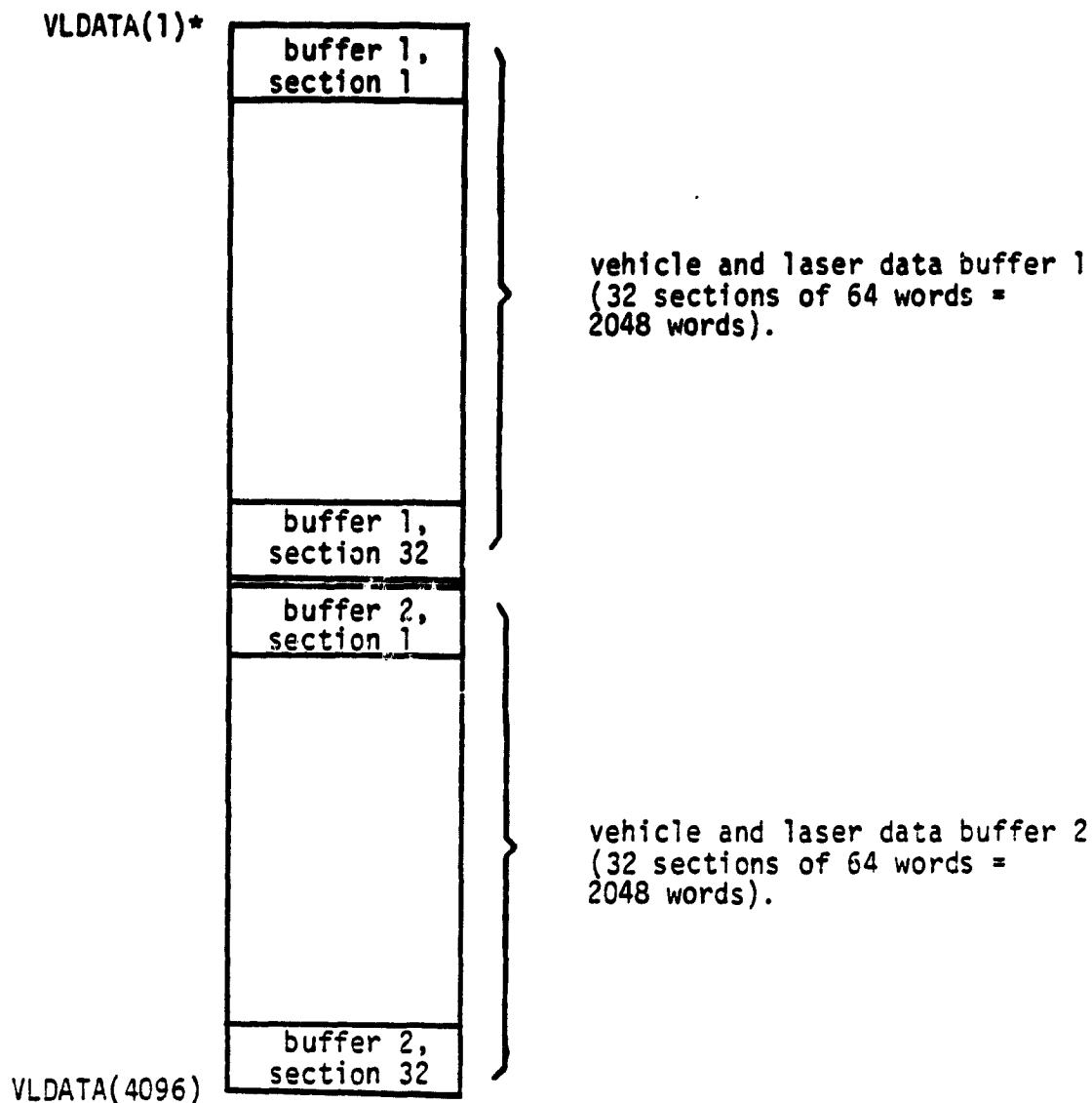
The buffers which the user specifies to be filled by T\$ROVR

must be referenced in the call to T\$ROVR which initializes it. The interrupt process of T\$ROVR fills two 2048-word buffers in the array named VLDATA. Each of these buffers is divided into 32 sections, one for each azimuth angle scanned by the laser mast. Each section contains 64 words consisting of 32 words of vehicle data followed by 32 words of laser data. The layout of each buffer is shown in Fig. 7.2.1.

The array VLSTAT contains status information which is stored there concerning each section of VLDATA as that section is being filled. The format of this array is shown in Fig. 7.2.2. The array ITSTAT contains status information about the GPIB interface itself. The information in ITSTAT can be updated by a special call to T\$ROVR, otherwise it is never changed. The format if ITSTAT is shown in Table 7.2.1.

It should be pointed out that the names assigned to the three arrays VLDATA, VLSTAT and ITSTAT used in the user's program are not important as long as the conventions for loading the programs are followed. These are explained in the documentation of T\$ROVR (Reference 5). All of the Fortran routines which run on the Prime use these names for the three arrays, however.

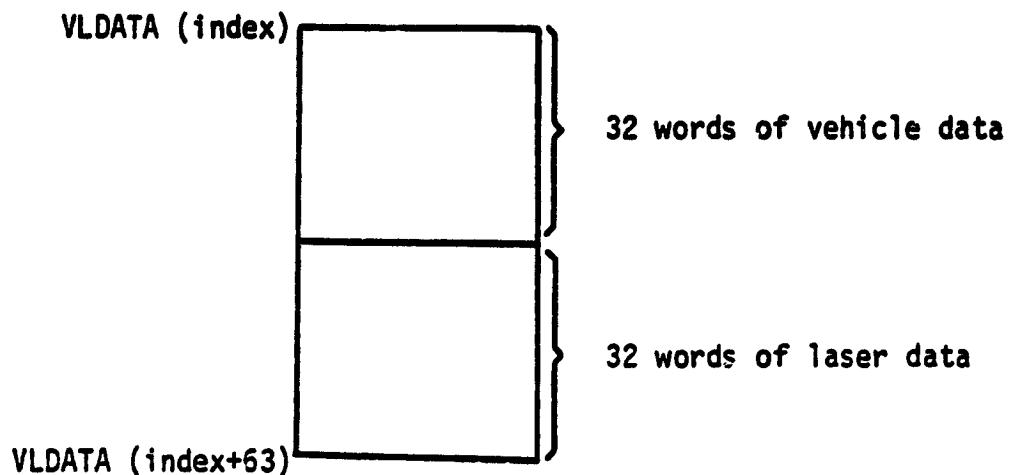
The scratch buffer used by T\$ROVR is needed in order for interrupts to be received even if both buffers in VLDATA are full. In general, it is bad practice to use this data in any calculations involving laser data because it will be constantly changing. If both VLDATA buffers are full, however, it is possible to get the



\* must be on a Prime memory page boundary

Fig. 7.2.1 Format of VLDATA array

A section of 64 words in VL DATA buffer (one of 32):



$\text{index} = 2048 * (\text{buffno}-1) + 64 * (\text{sectno}-1) + 1$

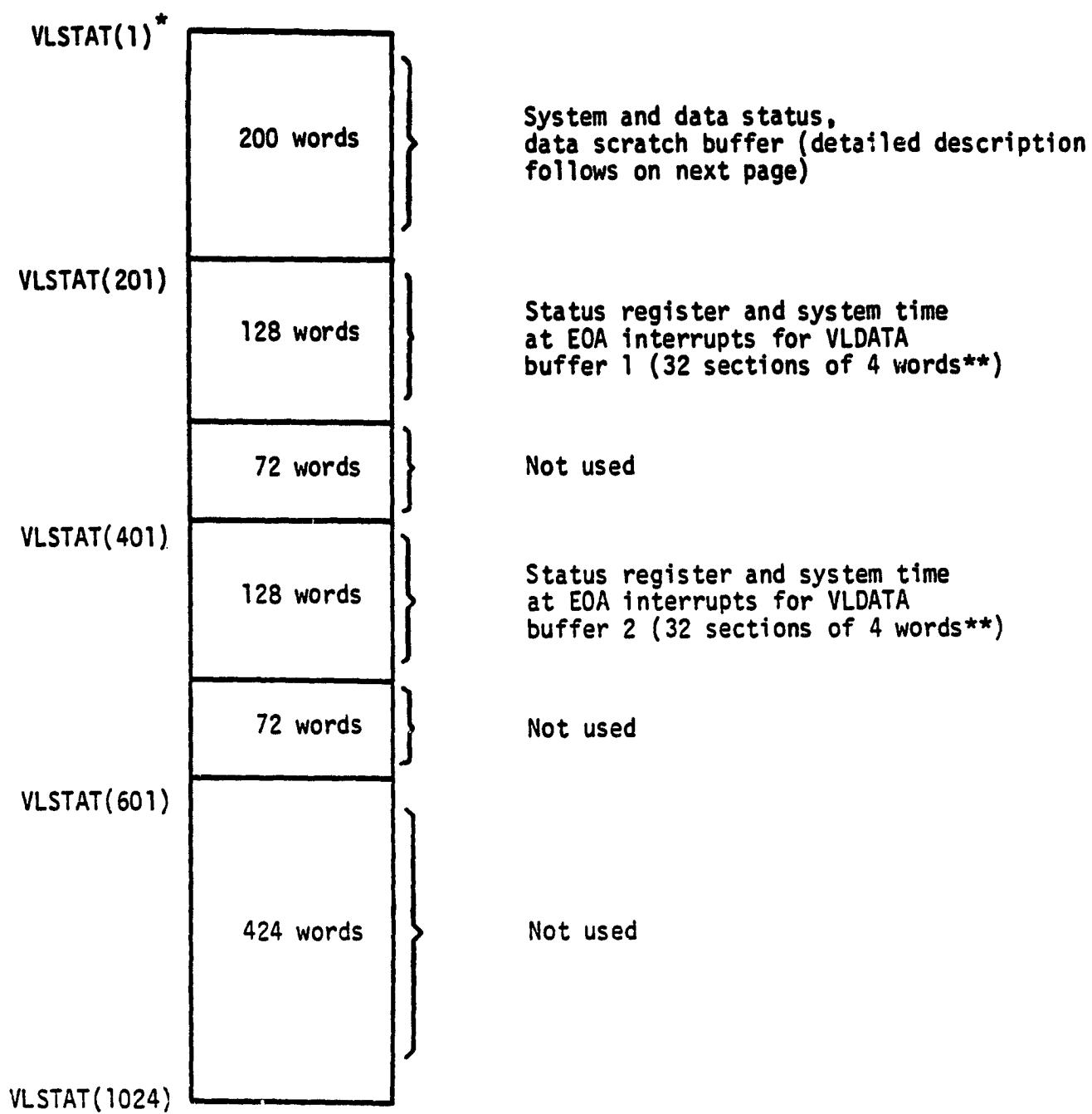
buffno . . . . buffer number (1 or 2)

sectno . . . . section number in one of the two buffers  
(1, 2, . . . , 31, 32)

#### Note

Each received data word is accompanied by an address word. The six least significant bits of the address word are used as offset into the current 64-word section and therefore determine where the data word will be stored within the current section.

Fig. 7.2.1 (continued)



\* must be on a Prime memory page boundary

\*\* 4 words per EOA interrupt:

word 1	GPIB status register
word 2}	
word 3}	Current system time
word 4}	(Format given in Fig. 5)

Fig. 7.2.2 Format of VLSTAT array

**System and data status:****VLSTAT(1)      System status:**

- 1 . . . system initialized, waiting for EOS interrupt to start scanning to VL DATA (data is currently received to data scratch buffer, rover is stopped)
- 0 . . . both VL DATA buffers are full (data is currently received to data scratch buffer, rover is stopped)
- 1 . . . data is currently received to VL DATA buffer 1, rover is started
- 2 . . . data is currently received to VL DATA buffer 2, rover is started
- 3 . . . received FIFO overflow while filling VL DATA buffer 1, waiting for EOS interrupt to start refilling buffer 1 (data is currently received to data scratch buffer, rover is stopped)
- 4 . . . received FIFO overflow while filling VL DATA buffer 2, waiting for EOS interrupt to start refilling buffer 2 (data is currently received to data scratch buffer, rover is stopped)
- 5 . . . received TIMEOUT error, waiting for EOS interrupt to start refilling current VL DATA buffer (data is currently received to data scratch buffer, rover is stopped)
- 6 . . . received raw error (no interrupt bits set), see Status register in VLSTAT(2), rover stopped, interrupts and DMX disabled, data not received.

**VLSTAT(2)**      CPIB Status register during the last interrupt, initialized to zero.

**VLSTAT(3)**  
**VLSTAT(4)**  
**VLSTAT(5)**      System time at the beginning of the last interrupt, initialized to zero. (Format given in Fig. 5)

**VLSTAT(6)**      Number of TIMEOUT interrupts\*  
**VLSTAT(7)**      Number of FIFO overflow interrupts\*  
**VLSTAT(8)**      Number of EOS interrupts\*

---

\* given as mod  $2^{15}$  number since system initialized.

VLSTAT(9)	Not used
VLSTAT(10)	Not used
VLSTAT(11)	Number of filled (valid) vehicle sections in VL DATA buffer 1 (0 . . . empty, 1-32 sections filled)
VLSTAT(12)	Number of EOF interrupts received while filling VL DATA buffer 1
VLSTAT(13)	Number of filled (valid) laser sections in VL DATA buffer 1 (0 . . . empty, 1-32 sections filled)
VLSTAT(14)	Number of EOA interrupts received while filling VL DATA buffer 1
VLSTAT(15)	EOS interrupt status of VL DATA buffer 1 0 . . . EOS not received 1 . . . EOS received, <u>VL DATA buffer 1 is full</u> , at this time VLSTAT(11) and VLSTAT(14) should contain 32
VLSTAT(16) to VLSTAT(20)	{ Not used
VLSTAT(21) to VLSTAT(30)	{ Same as VLSTAT(11) to VLSTAT(20) but apply to VL DATA buffer 2
VLSTAT(31) to VLSTAT(64)	{ Not used
VLSTAT(65) to VLSTAT(128)	{ Data scratch buffer, data is stored into this 64-word buffer while it cannot be stored into VL DATA buffers
VLSTAT(129) to VLSTAT(200)	{ Not used

Fig. 7.2.2 (continued)

ITSTAT(1)	Status register
ITSTAT(2)	Command register
ITSTAT(3)	DMX address register
ITSTAT(4)	Vector address and DMX mode register
ITSTAT(5)	ID slot register
ITSTAT(6)	Total number of interrupts generated by the interface since last initialize operation or last assign command
ITSTAT(7)	Number of "invalid address" interrupts (for T\$ROVR debugging only)
ITSTAT(8)	Number of "attempted page-fault" interrupts (for T\$ROVR debugging only)
ITSTAT(9)	Run flag: 0 . . . vehicle started 1 . . . vehicle stopped
ITSTAT(10)	VLDATA buffer 1 full flag: 0 . . . buffer empty 1 . . . buffer full
ITSTAT(11)	VLDATA buffer 2 full flag: 0 . . . buffer empty 1 . . . buffer full
ITSTAT(12)	System time (minutes)
ITSTAT(13)	System time (seconds)
ITSTAT(14)	System time (ticks, where 1 tick = 1/330 second)
ITSTAT(15) to ITSTAT(20)	{ Not used

Table 7.2.1 Format of ITSTAT array

latest vehicle data from the first part of that buffer due to the fact that the same analog and digital data will be stored at any particular location in the block of vehicle data. In the case of laser data received while both buffers are full, however, the user would have no way of knowing which azimuth was being stored in the scratch buffer when it was being read.

### 7.3 Debugging Aids for the GPIB Interface and Telemetry System

In addition to the I/O driver T\$ROVR, M. Potmesil also wrote a number of diagnostic programs that are useful for testing the interface between the telemetry system and the Prime. These programs are shown in Table 7.3.1. All of the programs in that table reside in the User File Directory (UFD) <SYSTEM0>TEST.GPIBS>TEST.T\$ROVR. There are also other programs, discussed below, which were written by the author. These programs are stored on the backup tape of the MARS UFD created at the end of the project and are located in the UFD named <USERS3>MARS>GRAHAM. Source listings are also in the notebook pertaining to the GPIB interface, volume 2.

The program named #SEND essentially has the same function as Potmesil's except that it tests all possible bit patterns which can be written into the command-transmit register on the GPIB interface. It also prints a message for every 1024 commands which are sent.

The program #SEND.GD can be used to send any of the currently implemented commands which the microprocessor of the laser mast recognizes. It was written primarily as an aid for testing

#START	Initializes T\$ROVR.
#STOP	Stops T\$ROVR from receiving data and interrupts from the GPIB.
#BADKEY	Calls T\$ROVR with a bad key and should generate an error message.
#BADBUF	Tries to initialize T\$ROVR with a bad buffer address for VLDATA. Note <u>both</u> VLDATA and VLSTAT must be located on a page boundary.
#STATUS	Calls T\$ROVR to get the GPIB status and prints it.
#SEND	Loads the command link register and reads it back to verify the contents of that register.
#SCAN	Fully tests T\$ROVR in diagnostic mode. This program generates simulated vehicle and laser data and inter- rupts and also demonstrates the correct use of T\$ROVR as a sub- routine called by Fortran programs.

Table 7.3.1 GPIB Diagnostic Programs.

actual commands sent from the Prime to the rover via the RF command link and also served as a reference for the software group when it came time for them to send commands in the path-selection and navigation routines.

The program #RECV.GD is probably the single most powerful tool in discovering problems occurring in the telemetry system. This program has the capability of displaying any data which T\$ROVR can store in the user's address space. It will empty the VL DATA buffers consecutively and can store any or all of the incoming data from the rover in these buffers as well as the contents of the VLSTAT and ITSTAT buffers on disk or magnetic tape for later use. In the early stages of the development of a running version of the controlling software on the Prime, this disk or magnetic tape data base could provide consistent data to that software. The program #RECV.GD has several interactive qualities which enable the user to selectively empty buffers, disable emptying buffers entirely, stop dumping data to disk and/or tape, and reinitialize T\$ROVR even in the middle of execution of the program, to mention a few of the possibilities. A complete list of commands for the program is contained in the GPIB notebook mentioned above.

There is also a MAP mode which will display a matrix of laser returns. This display condenses information about all azimuths and elevations at which laser shots occur into one screen image. A maximum of eight azimuths (columns) and 12 elevations (rows) can be displayed at any one time. The starting row and column of the

upper left-hand element in the matrix can be set by the NEnn and NAnn commands, respectively. The two numbers which appear at each location of the matrix are the identification numbers of the detectors which received returns from a given laser shot at the elevation and azimuth given by their location in the table. If asterisks appear at a location, then no data concerning a laser shot at that azimuth and elevation angle was received by the GPIB. If a location is blank, this indicates that a missing return occurred, i.e., none of the detectors received a return for that laser shot.

The program #POST.GD can be used to read the disk or magnetic tape files created by #RECV.GD and provides for displaying any of the data which T\$ROVR stored in the VLDATA, VLSTAT and ITSTAT buffers in a manner similar to #RECV.GD except that the data being displayed could have been received minutes or hours before. It features the same MAP mode as #RECV.GD and the commands which it accepts are also documented in the notebook along with the commands for #RECV.GD.

The programs in the UFD<USERS3>MARS>GRAHAM are of more interest to the hardware development group since they provide more data on the number of timeouts, total number of interrupts, et cetera, than the control-oriented software group would in general be interested in.

Several files in the UFD also contain code for the M6800 microprocessor which enable it to operate the telemetry transmitter

and send dummy data down to the Prime. Some of these are listed in Table 7.3.2. These programs can be downloaded into the microprocessor from the Prime by running the program \*DNLOAD and then connecting the microprocessor to the user's terminal as described in the notebook pertaining to the microprocessor (see Chapter 8).

<u>PROGRAM</u>	<u>FUNCTION</u>
XMIT.GD	Sends data from 1024 laser shots and 1024 vehicle words, and fills an entire VL DATA buffer. The address and data words are the same except for interrupt bits set for EOA, EOV and EOS interrupts. Data and address words increment from zero to \$7FF.
XMIT.NOEOV	Same as XMIT.GD except doesn't generate EOV interrupts. This program shows the dramatic effect EOV interrupts have on the system's performance as far as slowing it down.
XMIT.NOEOA	Same as XMIT.GD except doesn't generate EOV or EOA interrupts.
XMIT.NOINT	Same as XMIT.GD except doesn't generate any interrupts after the first EOS interrupt.

Table 7.3.2 Microprocessor Transmitter Programs.

PART 8  
LABORATORY REFERENCE MATERIAL

There are several notebooks in the lab which contain information about all parts of the systems controlling the rover. Each is labeled, making it easy to find information on any one of them. For example, the documentation on the GPIB interface takes up two such notebooks, Volume 1 containing mostly hardware specifications and Volume 2 containing mostly software specifications and listings. The titles of these notebooks are shown in Table 8.0.1.

<u>TITLE</u>	<u>CONTENTS</u>
MICRO	Contains documentation on the microprocessor board itself along with the instruction manuals supplied by Motorola and listings of all programs which were written for the M6800 for use by the rover project. Also has instructions for downloading programs from the Prime computer to the microprocessor.
PROPELLION/ STEERING	Contains the hardware description of the motor speed controller board as designed by both Turner and Bogdan, a copy of Turner's masters thesis, and lists of signals which the microprocessor uses and generates when it controls the vehicle.
TELEMETRY	Contains the hardware description of the telemetry transmitter and analog multiplexor boards as well as the final circuit diagrams of the command link.
GPIB, Vol. 1	Contains the hardware description of the circuitry on the GPIB which was designed by Donaldson. All modifications are noted.
GPIB, Vol. 2	Contains the listings and descriptions of programs which exercise the GPIB interface as well as those for use in debugging the telemetry data link.
MAST	Contains the hardware description of the mast electronics including the laser detector electronics.
REALTIME SOFTWARE	Contains documentation on the realtime programs which run on the Prime, the locations of mag tape backups for the Mars UFD, etc.

Table 8.0.1 Laboratory Notebooks

## PART 9

### REFERENCES

1. Turner, J., "A Propulsion and Steering Control System for the Mars Rover," Rensselaer Polytechnic Institute Report MP-77, Troy, N.Y., August 1980.
2. Cipolle, D., "A High Speed Telemetry Data Link for an Autonomous Roving Vehicle," Rensselaer Polytechnic Institute Technical Report MP-72, Troy, N.Y., August 1980.
3. Craig, J. and Yerazunis, S., "Elevation Scanning Laser/Multi-Sensor Hazard Detection System Controller and Mirror/Mast Speed Control Components," Rensselaer Polytechnic Institute Technical Report MP-59, Troy, N.Y., August 1978.
4. Donaldson, J., "Laser Optical Appraisal and Design of a Prime/Rover Interface," Rensselaer Polytechnic Institute Technical Report MP-68, Troy, N.Y., December 1979.
5. Potmesil, M., "T\$ROVR, A Primos I/O Driver for the Rover Vehicle," Rensselaer Polytechnic Institute, Troy, N.Y., July 1980.
6. Kennedy, W., "Control Electronics for a Multi-Laser/Multi-Detector," Rensselaer Polytechnic Institute Technical Report MP-73, Troy, N.Y., August 1980.
7. Odenthal, J., "A Receiver for Laser Triangulation Rangefinder," Rensselaer Polytechnic Institute Technical Report MP-74, Troy, N.Y., August 1980.
8. Bogdan, D., "A Propulsion System for the Mars Rover Vehicle," Rensselaer Polytechnic Institute Technical Report MP-70, Troy, N.Y., August 1980.
9. Coles, G., "Real-Time Operating System for a Multi-Laser/Multi-Detector System," Rensselaer Polytechnic Institute Technical Report MP-65, Troy, N.Y., July 1979.
10. Messing, B., "Interpretation of Laser/Multi-Sensor Data for Short Range Terrain Modeling and Hazard Detection," Rensselaer Polytechnic Institute Technical Report MP-75, Troy, N.Y., August 1980.