REAL - TIME DATA ACQUISITION FOR SURFACE MEASUREMENT

RESEARCH PROJECT # 1997-F

PROJECT TITLE:
IMPLEMENTATION OF INTELLIGENT BUS SYSTEM FOR DISTRESS MEASUREMENTS

THE UNIVERSITY OF TEXAS AT ARLINGTON
TRANSPORTATION INSTRUMENTATION LABORATORY

Research Supervisor: Roger S. Walker, Ph.D., P.E.
Serial No. 3154

November 1997

Notice *- The United States Government and the state of Texas do not endorse products or manufacturers. Trade or manufacturers' names appear solely because they are considered essential to the object of the report.*

Technical Report Documentation Page

1. Report No.	2. Government Accession No.	3. Recipient's Catalog No.
TX-97/1997-12		

4. Title and Subtitle	5. Report Date
REAL-TIME DATA ACQUISITION FOR SURFACE MEASUREMENT/IMPLEMENTATION OF INTELLIGENT BUS SYTEMS FOR DISTRESS MEASUREMENTS	November 97
	6. Performing Organization Code
7. Author(s)	8. Performing Organization Report No.
Roger S. Walker	Research Report 1997-F
9. Performing Organization Name and Address	10. Work Unit No. (TRAIS)
The University of Texas at Arlington Arlington, TX 76019	11. Contract or Grant No. Project No. 7-1997
12. Sponsoring Agency Name and Addres	13. Type of Report and Period Covered
Texas Department of Transportation Research and Technology Transfer Office P. O. Box 5080, Austin, TX 78763-5080	Final: September 93-August 97
	14. Sponsoring Agency Code

15. Supplementary Notes

Research perfomed in cooperation with the Texas Department of Transportation.

16. Abstract

This report provides specific details on much of the work done on a project for the Texas Department of Transportation for providing the capability for making pavement distress measurements. The report primarily contains hardware and design procedures used to implement the Texas Profiler/Rut-Bar systems and which will be useful for TxDOT personnel in using and maintaining this equipment.

17. Key Word	1 B. Distribution Statement
TxDOT Profiler, Rut Measurements, Real-time, Pavement Distress Measurements	No restrictions. This document is available to the public through the National Technical Information Service, Springfield, Virginia 22161.

19. Security ClassiF. (of this report)	20. Security Classif. (of this page)	21. No. of Page	22. Price
Unclassified	Unclassified	232	

Form DOT F 1700.7 18-72)

IMPLEMENTATION STATEMENT

The Pavement Section of the Texas Department of Transportation has recently constructed and is currently implementing a number of profiler/rut bar vans. Much of the technology and procedures for these systems are a direct result of the research performed in this project. The system concepts developed during this implementation phase were designed so each vehicle will be capable of collecting a variety of pavement surface distress data. The successful completion of this project has provided a more accurate and quicker method of obtaining various distress information for the State's PES data base and for project specific applications.

DISCLAIMER(S)

The contents of this report reflect the views of the author(s), who is (are) responsible for the facts and the accuracy of the data presented herein. The contents do not necessarily reflect the official view or policies of the Texas Department of Transportation. This report does not constitute a standard, specification, or regulation.

There was no invention or discovery conceived or first actually reduced to practice in the course of or under this contract, including any art, method, process, machine, manufacture, design or composition of matter, or any new useful improvement thereof, or any variety of plant, which is or may be patentable under the patent laws of the United States of America or any foreign country.

ACKNOWLEDGMENTS

This work was a team effort with Wen-Ming Kuo, Brian Burgess, and as a graduate student, Guor-Chaur Jung, was especially instrumental in the team effort as well as other graduate students who worked on the project at various times.

Acknowledgement should be given to Carl Bertrand, Project Director, of the Texas Department of Transportation. His vision of an up-to-date fleet of surface measuring equipment has provided TxDOT with the latest advances in measurement technology. Also, acknowledgement should be given to Mr. Robert Light who provided valuable input into the measurement equipment and processes.

TABLE OF CONTENTS

TABLE OF FIGURES

List of Tables

CHAPTER 1
Introduction

This report provides specific details on much of the work done on a project for the Texas Department of Transportation for providing the capability for making pavement distress measurements. The original objective of the project was stated as:

> The Texas Department of Transportation must collect different types of data for both project and network level applications. The data is obtained from several different instruments and sensor types which are often housed in different vehicles. The data collection process thus involves the use of many different operators in different vehicles. Because of the different equipment types multiple passes over the same surface are often required. The data collection and processing procedures involve the Surface Dynamics Profilometer, the Siometer, and the Automated Road Analyzer or ARAN vehicles. The ARAN vehicle has been converted and updated from its' original configuration and soon will provide not only video logs of pavement surfaces, rutting information, and ride data, but also, surface profile and pavement cracking.
>
> With so many different applications in the pavements' field, it is becoming more and more desirable to integrate all of these operations into one data collection system. The system needs to handle not only today's real-time requirements, but also those of future applications and needs.
>
> In the project, "Real-Time Bus System for Interface of Surface Measuring Instruments", Study No. 1932, a real-time bus design has been developed which can provide such interface. A project is now needed for the implementation of this bus system and its usage in the Department's distress measurement vans, which are being developed by the Pavement Section.

A common task in this and other projects of a similar nature is to work with the rapid changes in technology used in implementing the concepts developed during this project. Since the first introduction of the PC by IBM in 1981 there has been an explosion in the development of PC compatible systems, first in desk top and notebook PC's and now in small modular boards running DOS or Windows CE. This technology is now being used in many instrumentation applications. Thus, much of the equipment initially considered for implementing this research is already outdated. The project has attempted to work with these rapid changes by developing modular concepts "when possible" which will permit system upgrade and still work within the original project objectives.

1.0 Report Contents

The report primarily contains hardware and design procedures used to implement the Texas Profiler/Rut-Bar systems and which will be useful for TxDOT personnel in using and maintaining this equipment. The next chapter provides details on the overall system concept, followed by details on the hardware modules used in the measuring system. Schematics, hardware design criteria, pal equations, and other design details are provided. Documentation of TALK, the Siometer Rut Bar communication program is provided in the Appendix.

CHAPTER 2
Systems Concepts

2.0 Measurement System

As noted in the previous chapter, the initial project objective was to develop and then implement a system for distress measurements which could be integrated with other systems, such as the video recording and data base system (developed by C Map Systems). As also noted, because of the rapidly changing technology, a modular approach was desired. Although initially, pavement profile, rutting, IRI and PSI measurements were desired, the system needed to be expandable to include other future measurements. Texture, for instance, is planned for implementation in 1998. The original plan included the use of the existing Siometers which computed SI and rut, in conjunction with general purpose 68020 boards linked using the VME bus. Based on this plan, the project personnel developed a reflective memory concept which could interface with various and different modules. Later, it was decided to simply use an existing 68020 board (sometimes referred to as a 'K' board) which had much of the initial desired processing capability. With the success in using the 68020 board, the Siometers were phased out. The project then developed specific boards, such as the Smart A/D, etc., which could then easily work with either this board or ones implementing the reflective memory concept. The reflective memory concept is described in the next section as it may yet be useful in later implementations. A wire wrap version of the board was done and initial printed circuit board (PCB) considered.

The description of the reflective memory board, and the other modules included in this report, are useful so that the Department can easily use these modules, or obtain additional modules as new requirements occur.

The complete measurement system described in this report include the following modules: two laser interface modules, a signal interface module, a laser power module, a smart A/D module, and the 68020 module. The reflective memory will be described in this section. The other modules are described in the chapters which follow.

2.1 Reflective Memory Overview

This section provides operating instructions and general information for the use of the Reflective Memory Board (RMB). The purpose of the RMB is to provide high speed data transfers between the VME type modules and Siometer Computer boards The original Siometer was a self-contained processing module, with no need to communicate with other processing units.

A previous solution to integrating various 68020 modules with the Siometer utilized a parallel interface. This parallel interface was implemented with a Motorola 680230 Parallel Interface/Timer on each. The ideal solution would be to have all required processors on the same board. The processors could then communicate with each other via a multi-ported RAM. In this way, the processors would not be required to be interrupt driven. With a simple message passing scheme, large amounts of data may be passed between any of the processors. One processor could read the data from a sensor, and then pass this data to all of

the other processors simultaneously. This, of course, is the ideal solution, but is not feasible for two reasons. First, is board space. The board to implement all of the processors would be much too large. Furthermore, where would one get say, quad-ported RAM? This idea does help, because it gives the best possible throughput. Each processor could communicate over a shared memory space. Communication is handled just like any read or write to memory.

The idea of using multiple dual-port RAMs, with an additional processor handling the traffic between them evolved into the reflective memory board concept. In this way, any data written into one would be "reflected" in the others. The reflective memory concept is illustrated in Figure 2.1.

2.1.1 Reflective Memory Board

The RMB provides 1024 bytes of reflective memory (one Kilo-byte). The RMB may interface up to four (4) processing boards. Table 2.1 lists all of the signals required by RMB.

Signal Name	I/O
A0-A9	Input
D0-D7	Both
CS*	Input
R/W*	Input
DTACK	Output

Table 2.1 RMB Signal List

The RMB consists of eight (8) FIFOs, two for each processor interface. Each FIFO is nine (9) bits wide. This allows the capture of the ten (10) address lines and eight (8) data lines. The ten (10) data lines allow the addressing of one Kilo-byte of memory. The chip select signal (CS*) is active low, and indicates that the processor interface is requesting a bus cycle to the RMB. The read signal (R/W*) indicates the direction of the transfer. If the transfer is a write, then the RMB writes both the address and data lines into the two (2) associated FIFOs. After writing the address and data into the FIFOs, the RMB asserts DTACK until the chip select is negated. If the bus cycle is a read, then the RMB reads the data from the dual-ports address, as specified by the address lines. The data from the dual-port is then placed on the processor's data lines, and DTACK is asserted to indicate that the data is valid. When the processor negates the chip select, the RMB tristates the data lines, and negates DTACK. It should be noted that the MACH 130 monitors and controls all of the FIFO and dual-port memory control signals. The RMB as a whole, appears just like a memory device to the processor board.

The following is an example of the steps taken by the RMB for each of the possible bus cycles. For the first case, assume that a processor board wants to write data to the RMB. It will set the R/W* signal to a logic low, to indicate a bus write. It then asserts its' chip select signal. The RMB detects the chip select and examines the direction of the bus cycle, which is a write in this case. It then ensures that the associated FIFOs are not already full. If the FIFOs are not full, it performs a write to both simultaneously. This write to the FIFOs,

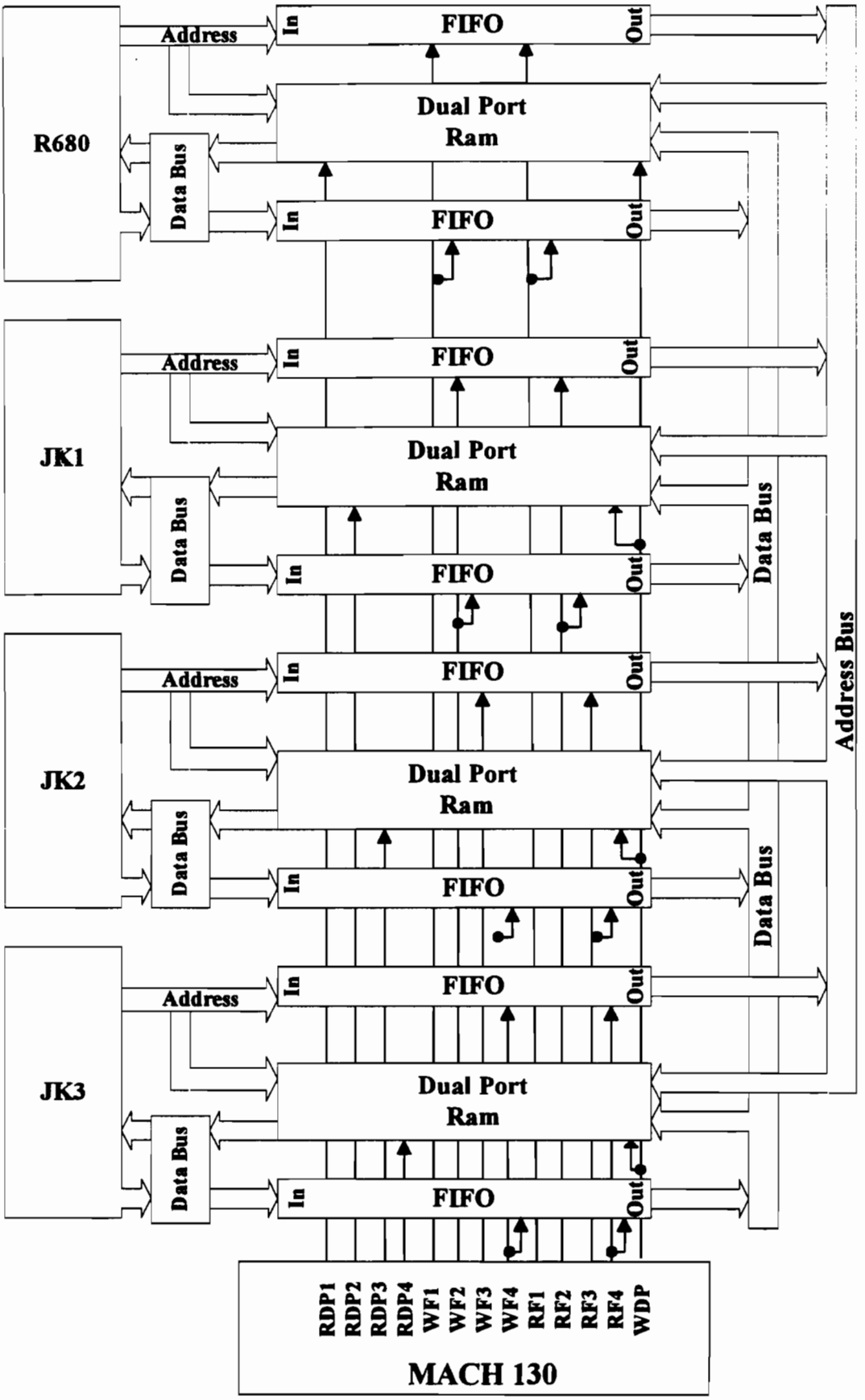


Figure 2.1 Reflective Memory Concept

captures both the address being written to, as well as the data. DTACK is then asserted to indicate that the bus write has been finished. All of the steps taken so far have been handled by one of the four (4) state machines (one for each possible processor board) that monitors the chip select signals from the processor boards. It is now ready for another bus cycle from the processor. At the point where the data was written into the FIFO, another state machine detected that the FIFO was not empty. This state machine now performs a read on the RMB side of both FIFOs. The address that was written is placed on the RMB's address bus, which is routed to the RMB's side of all the dual-ports (examine the diagram in Appendix A). At the same time, the data (as written by the processor board) is routed to the data lines on all the dual-port RAM. After the appropriate set time has been met, the RMB performs a write to all of the dual-port RAMs simultaneously. The data that was written by the one processor board has now been placed in the appropriate address in all of the dual-port RAMs.

Now, the other case is examined. Assume that the data just written by the one processor board is to be read by another. This processor board performs a bus read. The new processor board sets the R/W* signal to a logic one, and asserts its' chip select. The state machine in RMB again detects the chip select, and examines the state of the R/W* signal. This time, it is read. Note that the address and data lines from the processor board are connected to both of the FIFOs, as well as the processor boards side of its' associated dual-port RAM . The state machine now performs a read from the processor boards side of the dual-port RAM. The correct data is read from the dual-port RAM, and placed on the processors data bus. DTACK is again asserted to signify the completion of the bus cycle.

The above example is simplified somewhat. Other conditions that the different state machines must account for are the busy signals from the dual-port RAMs, full signals from the FIFOs, which FIFO to read from if more than one has data (not empty), and the timing of the DTACK signal when a bus cycle is extended for one of the above conditions.

A brief description will now be given for each of the different state machines on the RMB. The first state machine is the interface state machine. It simply waits for the chip select signal from its' associated processor board. When the chip select is asserted, the state machine either writes the data and address into the two associated FIFOs, or reads data from that processors dual-port . This depends on the direction of the bus cycle as indicated by the R/W* signal. The interface state machine is implemented four (4) times in the MACH 130. Each one runs independently of the others, and is dedicated to a particular processor board.

The other state machine, has the responsibility to read the data from all of the FIFOs, and place it concurrently into all of the dual-port RAMs. This state machine is termed "RMB Controller State Machine". The RMB state machine is really a combination of two state machines. The combination comes from the fact that more than one FIFO may contain data that needs to be routed to the dual-ports. Because of this possibility, some sort of scheduling is required. The scheme implemented on the RMB is a simple Round Robin technique. This guarantees that all of the FIFOs will be read, even if they all contain data. To implement the scheduling, a small state machine called the scheduling state machine, keeps track of which FIFOs contain data, and which FIFO was previously read. Using this information, the scheduling state machine indicates to the RMB controller state machine which FIFO to read.

CHAPTER 3
PC-68020 Module Communication

3.0 Procedures

Three types of programs are used for making pavement distress measurements with the measurement system described in this report. The 68020 module is also often referred to as the K board. One is the basic K board program (JKROM). This firmware program is in EPROM and used to interface with the PC. It can be thought of a program similar to BIOS in a PC, as it is initiated when power on or a reset is applied to the K processing board and begins the initialization procedures. It also provides the I/O drivers for the various sensors which interface with the K board, as well as, performs all real-time profile and rut processing done in the K board. The second program, TK, runs on the PC and is used to communicate with the K board program JKROM, and the PC. The third class consists of various post processing software for computing PSI, IRI, and Rut, and to display the data obtained from TK. This chapter will discuss the communication protocols which are used by TK for communicating with the K board via JKROM. This information is necessary for communication with the K board system.

3.1 General 68020 K board Firmware Functions

The 68020 processor module provides a wide selection of data collection and measuring options. The board or module, when inserted in a suitable environment, such as described in this report provides profile and rut for pavement management. The board uses an RS232 compatible serial interface for communication with and data storage to an IBM compatible PC (desktop, laptop or notebook), where profile or other roughness information can be sent in real-time for data storage and later analysis. The system can be installed in most automobiles or vans for various surface measurements.

Profile measurements are accomplished by using a modified form of the South Dakota method with an accelerometer and acoustic or laser sensor. In this method, an accelerometer (sampled and integrated with respect to time) and laser sensor are used for measuring body and road-body displacements. Road profile measurements are obtained by summing the twice integrated acceleration measurements with the appropriate body-road displacements. The R68020 board in conjunction with the PC can provide profile and rut measurements using the European string line method. Real-time IRI can be provided by using a second board, although it is no longer needed with the speeds of the PC. PC software can compute IRI in real-time from the profile data which is being sent from the 68020 module. The next section describes details on the K board to PC communications. The last section describes the real-time rut capability.

3.2 K Board PC Communication

The K board - PC Communications details which follow, reference the J processor board. In the initial systems, real-time data acquisition and processing was done by two

independent processors, the J and K processors. Later, it was determined that the needed processing capability for the data measurement process could be accomplished by a single processor (K Board) and PC. The communication protocol descriptions in this section include communications with both the J and K board processors.

3.2.1 Overview

The PC, the J processor, and the K processor are the computing units of the system. The PC controls J and K in the data collection session and provides storage for programs, configuration data, and acquired road data. The PC can be used to compute IRI in real time by using the profile data from K. The K processor reads the sensors, computes profile and rut and sends the data to the PC. The K processor may also send sensor data or profile to the J Processor, receive the IRI data from J, and relay the IRI data to the PC. The J processor receives sensor data or profile data from K, computes IRI and sends it to K. The serial port A of K is connected to the COM1 or COM2 of the PC. The serial port A of the J processor is connected to the serial port B of the K processor. All the communications between J and the PC must go through K.

3.2.2 Definitions

The J and K operate in two modes: the *monitor mode* and the *application mode*. After power-up or reset, they are in the monitor mode. The monitor mode is only used to download the application program to the RAM and start running it. Program download is not necessary if the current program resides in the ROM. Once the application program starts running, J/K will be in the application mode until a hardware reset.

The commands sent to J/K will be expressed as

a quoted string: if the command is composed of all displayable characters,
a symbol: if the command is a non-displayable character.

The carriage return is expressed as <CR>, and the escape character as <ESC>. A symbol is usually followed by its hexadecimal value in parentheses. The hexadecimal number is preceded with 0x. The plus sign + is used to concatenate commands. No plus signs or quotes should be sent as part of the commands.

When J/K is in the monitor mode, each command is a string followed by a carriage return. When K is in the application mode, the PC sends a one-byte command to K to specify an option, to initiate a sequence of actions, or to terminate an action.

There are various types of data packets passing between K and PC and between J and K. The data packet always begins with a control byte followed by a number of data bytes. The most significant bit (#7) is always one in the control byte and is always zero in the data bytes so that the beginning of the packet can be easily identified.

Since we are using only seven bits in a data byte, the binary data that takes more than seven bits needs to be encoded for transmission.

A four-byte floating point number is encoded into five bytes:
1st byte: bits 0-6 of the floating point number,
2nd byte: bits 7-13 of the floating point number,
3rd byte: bits 14-20 of the floating point number,
4th byte: bits 21-27 of the floating point number,
5th byte: bits 28-31 of the floating point number.

The symbol *GetFloat* will be used in this document to specify the procedure of reading five bytes from the serial port to construct a floating point number.

Similarly, *GetLong* will be used to denote reading five bytes to construct a 32-bit integer, *GetByte* for a 7-bit integer; *Get14* for a 14-bit integer, *Get21* for a 21-bit integer. The receiver of the data should interpret the integer as signed or unsigned as implied by the packet. For example, the time between acceleration samples is unsigned, while the profile is signed.

A string will be sent as is with a null character at the end. The symbol *GetString* will be used for the procedure of reading a string from the serial port until a null character is encountered.

The *scale factor* and *offset* are used to convert A/D converter reading to real units.

laser data in mm = laser A/D value * laser scale factor + laser offset
acceleration in mm/sec^2 = acceleration A/D value * acceleration scale + acceleration offset.

The *time between acceleration samples* received from K is in the unit of the timer ticks used in the system. This value should be multiplied by the *time scale factor* to give time in seconds. The scale factors and offsets will be sent to the PC when the PC initiates data collection.

3.2.3 Starting Application

To download the K application program to the RAM,

1. make sure K is in monitor mode;
2. send "L<RET>" to enter S-record download mode;
3. send the S-record of the K application program.

To run the K application program,

1. send "g 7000<CR>", if the program is in ROM; or
2. send "g 80002000<CR>", if the program is in RAM.

To initialize K

1. send JKC_INI300 (0xD5) to initiate data transfer;
2. send the entire content of the file K3000000.INI;
3. send JKC_ENDINI (0xC4) to terminate data transfer.

To download J application program to the RAM,

1. make sure K is in application mode and J is in monitor mode;
2. send JKC_TOGMONITOR (0xD0) , so that the subsequent data will be passed on to J
3. end "L<RET>" to enter S-record download mode
4. send the J program S-record
5. send JKC_TOGMONITOR (0xD0) to stop passing data to J.

To run the J application program,

1. make sure K is in application mode and J is in monitor mode;
2. send JKC_TOGMONITOR (0xD0) , so that the subsequent data will be passed on to J;
4. send "g D000<CR>", if the program is in ROM; or send "g 80002000<CR>", if the program is in RAM;
5. send JKC_TOGMONITOR (0xD0) to stop K from passing data to J

To initialize J,

1. make sure both J and K are in application mode;
2. send JKC_TOGMONITOR (0xD0) , so that the subsequent data will be passed on to J;
3. send JKC_INI (0xC3) to initiate data transfer;
4. send the entire content of the file J0000000.INI;
5. send JKC_ENDINI (0xC4) terminate data transfer;
6. send JKC_TOGMONITOR (0xD0) to stop K from passing data to J.

3.2.4 Sending Data Collection Options

All data collection options must be sent to K before starting data collection:

To collect string-line rut,
send JKC_WANTRUT (0xCA).
To collect vertical displacement data (raw rut data),
send JKC_WANTRUTRAW (0xCD).
To get the list of active lasers,
send 'a' or JKD_ACTIVE (0x8C).

The string-line and raw-data options are mutually exclusive. It is recommended to get the list of the active lasers when collecting raw data.

To collect raw profile data (acceleration, vertical displacement, and time between samples),
send JKC_WANTACC (0xC9).

To collect profile,
send JKC_WANTKLEFT (0xCC) for left profile only;
send JKC_WANTKRIGHT (0xCF) for right profile only;
or send JKC_WANKLEFT + JKC_WANTKRIGHT for both left and right profiles.

To collect one-wheel-path IRI,
send JKC_WANTIRI (0xCB).

To collect two-wheel-path IRI,
send JKC_WANT2IRI (0xD1).

The above four profile options, raw data, profiles, one-wheel-path-IRI, and two-wheel-path-IRI, are mutually exclusive.

To specify timer frequency (only useful when the time/distance switch is switched to time), send JKD_SIMUSPEED + frequency in Hz in the ASCII-encoded real number format + null character.

3.2.5 Starting Data Collection

To enable serial output,
send JKC_SERIAL (0xC7) or 's'.

To start pre-section (always),
send 'p' or 'P'.

To arm the section mark detector,
send JKC_ARMREALMARK (0xD4).

To start real section,
send 'r' or 'R'.

Arming the section mark detector allows the K processor to send JKC_REAL (0xC1) to the PC. The PC shall always inform K to start real section. In the case where pre-section data is not needed, the PC shall send 'r' immediately after 'p'.

3.2.6 Reading K Processor Output

Upon receiving JKC_INI (0xC3, as a result of starting the presection),

1. call GetByte to receive the flag that indicates the unit system;
   (Currently the value is always non-zero, indicating the metric system.)
2. call GetFloat to get the acceleration sampling interval in meter;
3. call GetFloat to get the average rut distance in meters;
4. call GetFloat to get the *laser scale factor*;
5. call GetFloat to get the *laser offset*;
6. call GetFloat to get the *acceleration scale factor*;
7. call GetFloat to get the *acceleration offset*;
8. call GetFloat to get the *time scale factor*.

Upon receiving JKD_ACTIVE (0x8C), as a result of requesting active lasers, call GetByte 16 times to construct an array of 16 flags.

Each byte indicates whether the corresponding laser channel is active (0: not active, non-0: active). The *total number of active lasers* shall be used in controlling the number of repetition in reading raw rut data.

Upon receiving KD_PROFILE (0x83), as a result of requesting one-wheel-path profile,

call Get21 to get the one-wheel-path, signed profile value.

The unit of profile is entered via the configuration program before the data collection. If the profile value is 0x100000 before the sign-extension operation, it's a bad profile value. This also applies to two-wheel-path profile too.

Upon receiving JKD_KPROFL (0x90), as a result of requesting two-wheel-path profile,

1. call Get21 to get the left profile;
2. for K program dated before 02/96, read one byte JKD_KPROFR (0x91) before reading right profile;
3. call Get21 to get the right profile.

Upon receiving JKD_ACCEL16 (0xA4), as a result of using 16-bit A/D converter and requesting raw profile data,

1. call GetLong to get a 4-byte integer;
2. get the signed A/D value for the accelerometer from the upper 16-bits;
3. get the signed A/D value for the laser from the lower 16-bits.
4. call Get21 to get the unsigned time value between samples.
5. If the time value is TF21_TOOLONG, the time between samples is too long.
6. If the time value is TF21_NOACC, the acceleration is invalid.
7. If the time value is TF21_NOLASER, the laser value is invalid.

Upon receiving JKD_ACCEL (0x85), as a result of using 12-bit A/D converter and requesting raw profile data,

1. call Get14 to get the unsigned accelerometer A/D data;
2. call Get14 to get the unsigned accelerometer A/D data;
3. call Get21 to get the unsigned time value between samples.
4. check time value following the previous rules.

When collecting one-wheel-path IRI, upon receiving JKD_IRI (0x81),

1. call GetFloat to get the IRI;
2. call GetFloat to get the average speed in km/hr;
3. call GetFloat to get distance since data collection in km.

Upon receiving JKC_REAL (0xC1),
K has detected the section marker; it's time to send 'r' to K to start real section.

Upon receiving JKD_ERROR (0x8F),

call GetString to get an error message from K.

3.2.7 Terminating Data Collection

Send <ESC> to terminate data collection.

3.3 Rut Measure Procedures

A description of the rut bar procedures developed for the Siometer/rutbar was developed and discussed in Research Report 1290-1F 6-1995. The PC computes the rutting statistics that are used for PMIS. The 68020 module described in this report, can directly compute the general string line rut statistics in real-time for laser sensors. This section will discuss the general string line method which is computed in real-time for the 68020 module.

The string line method is based on five displacement measuring sensors placed on the front of the automobile as illustrated in Figure 3.2.1. The Texas DOT has developed rut bars for each of the vans used for rut and roughness measurements using acoustic sensors. TxDOT is currently constructing a five laser rut bar for evaluation in replacing the five acoustic sensors with lasers. Two of the lasers will be used jointly for both rut and profile calculation.

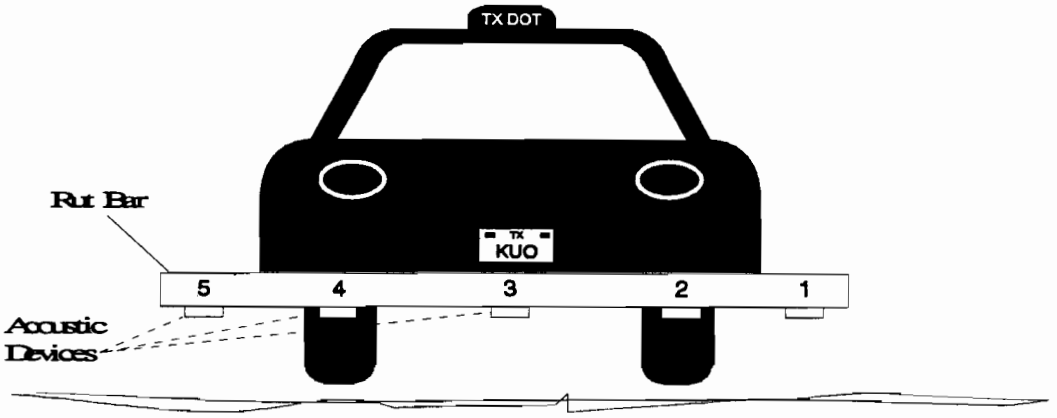


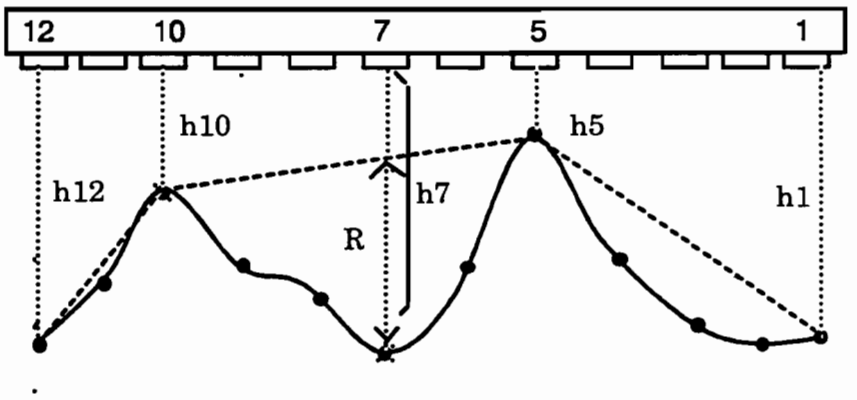
Figure 3.1 Texas Five Sensor System Rut Bar

3.3.1 String Line General

The various rut methods used by TxDOT may be considered variations of the general string line procedure which is illustrated in the Figure 3.2. This general method is outlined as follows:

If we lay a hypothetical string along the cross section of the surface of road, all the straight or convex portions of the road will contact the string, and all the concave portions of the cross section, the ruts, will be under the string. The position of the deepest rut is found where the surface of the road is farthest from the string. This maximum depth is defined as the rut depth of the cross section. Practically, we use distance sensors to measure the depths of a number of nodes on the cross section with respect to a horizontal rutbar. The measured cross section of the pavement is simplified to a polyline. The hypothetical string also becomes a polyline. Thus the nodes on the cross-section polyline that do not touch the string are the rut candidates. An algorithm was developed for the computer to construct this string and find the nodes that are ruts. The general method is illustrated in the figures following.

2. String Line General

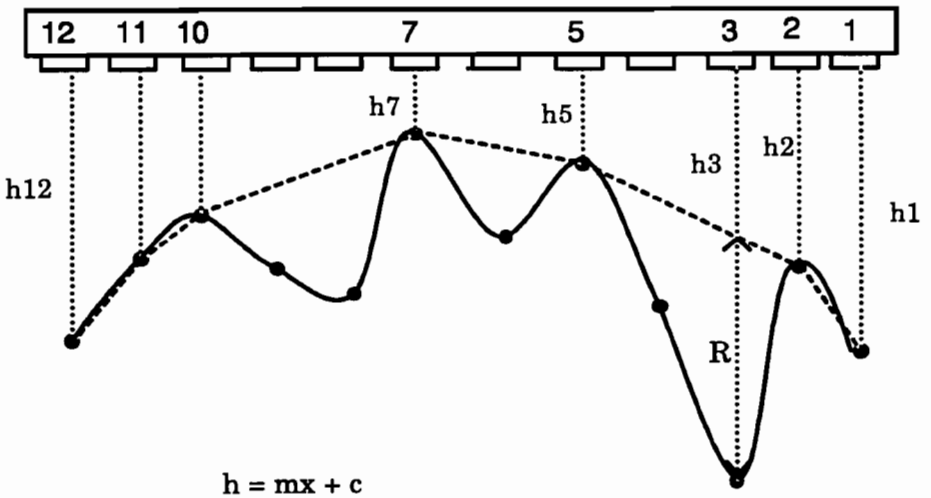


h = m * x + c

m (slope) = (h10 - h5) / (x10-x5)

c = h10 - (m * x10)

R = h7 - (m * x7 + c)



h = mx + c

m (slope) = (h5 - h2) / (x5-x2)

c = h5 - (m * x5)

R = h3 - (m * x3 + c)

Figure 3.2 String Line General

CHAPTER 4
Main Chassis Module

4.0 Overview

The main chassis module contains the various system components for computing profile and rut. It interfaces with the PC and system sensors. Either real-time or post processed profile can be obtained for two independent wheel paths. The rut measuring system, likewise can provide rut data for using up to 10 acoustic channels, or 5 channels using the real-time Texas Rut procedure as noted in Chapter 3. The Texas rut method uses the PC for the real-time rut computations. The raw read readings may optionally be saved and later post processed. The 68020 processor module has the capability for computing real-time laser rut for up to 11 lasers using the string line method, also described in Chapter 3. However, in order to facilitate laser rut, the main chassis module would need to be modified to interface with the additional lasers. A functional description of each module follows:

4.1 Functional

A basic layout of main chassis unit is illustrated in Figure 4.1. As illustrated in the figure, the unit consists of the 68020 module or K board, three power supplies, two laser interface modules, a signal interface module, a laser power module, and a smart A/D module. The unit receives signals from two accelerometers, two Selcom lasers, a distance sensor and an event or infrared start sensor. The front control panel, Figure 4.2, provides controls for turning on the unit, resetting the processor, and for selecting either the distance signal for normal operations of a simulated distance signal (time/distance switch) for testing. The back panel, Figure 4.3, provides the connectors for the various sensors, interface to the PC, power and fuse connectors.

The three power modules provide power for the interface and processor boards, as well as the two Selcom lasers. Two 24 volt power supplies provide positive and negative voltages to the laser power modules (Chapter 5). This module then provides the necessary voltages for the lasers, signal interface module, smart A/D module, and laser interface modules. There have been two slightly different versions of the main chassis module. In the first one, the third power supply provides a +5, and ± 12 voltage. The 5 volt is used for the digital logic. The ±12 volts are used primarily for the operational amplifiers as will be illustrated in the following power wiring diagrams. Twelve volts are also sent to the cooling fan and for power to the two lasers. In the second version, a single linear +12 volts is provided. The required digital +5 volts is supplied by two three amp five volt regulators, one for the 68020 processor module, which draws the most current (1.5 amps), and the second for the remaining 5 volt requirements. Twelve volts is also sent to the cooling fan and for power to the two lasers. In this second version, the ±15 volts from the laser power module are used for providing power to the operational amplifiers. The specifications for the 24 volt SOLA supply is given in Table 4.1 and the multi-voltage switching Condor power supply used for first version, in Table 4.2. The 12 volt single SOLA supply used in the second version is similar to Table 4.1, except the 12 volts is rated at 5 amps.

The processor module, which is further described in Chapter 9, provides the computational capability for the profile and rut calculations, controls the overall system operation and provides communication with the PC and operator. The processor module is wire wrapped on a standard VME compatible Euro card and connects to the other modules via the two 96 pin connectors, J1 and J2. The specific pin assignments are provided in Chapter 10. Compatibility with signals (power and ground) were made for the J1 and J2 connectors on the VME bus. The VME bus J1 pin outs are specified in Table 4.3 for reference. The J2 rows A and C are available for general use as will be illustrated in Chapter 9.

The two accelerometer signals are received via two BNC connectors (Table 4.6), where the center pin is used for the ±5 volt signal. Each of the analog accelerometer voltages, which are proportional to the vehicle acceleration as sensed by the two accelerometers located next to each of the two lasers, are received by two independent low pass filters. The 400 hertz filters are used to prevent aliasing as the signals are sampled via the smart A/D module at 1000 hertz. This is shown in Figure 4.4 (a).

The two lasers, which are used to provide the road body displacement, connect to the unit via two Amphenol 10 pin connectors. Table 4.5 provides the pin out used. The connections provide for both signal input as well as power for the lasers. Each laser has a separate laser interface module (Chapter 7) which converts the serial data stream (differential data and clock signals) from the Selcom laser to an analog signal. The modules convert the serial data to a positive 0 to 10 volt signal, whose amplitude is proportional to the distance from the laser to the object to be measured. The analog laser signal is sent to a buffer on the signal interface module and then on to the smart A/D module. This is shown in Figure 4.4 (b).

The signal interface module also receives signals from the distance sensor and infrared start signals via two BNC connectors (center pin). Each signal is first optically isolated. The distance signal, is passed through a one shot and optionally, a divide circuit, and then sent on to the time/distance switch on the front control panel. The switch is used to select either the actual distance signal or a simulated distance signal. Either this signal, or the simulated distance signal generated by one of the timers on the processor module is then sent to the distance input on the processor module. The start signal, after it is isolated, is converted to a TTL signal for input to the processor module for initiating the profile and and/or rut measuring process. These two signals are shown in Figure 4.4 (c).

Rut measurements are obtained through the acoustic sensors. The displacement of the rut bar with respect to the pavement is accomplished with a signal pulse, where the signal width is proportional to the distance. Two separate Amphenol 7 pin connectors are used for the acoustic sensors one to five, (Table 4.7) and sensors six to ten, (Table 4.8). The five Texas rut system uses sensors one to five.

The personal computer or PC interface is connected via a three wire serial (RS232 compatible) 25 pin DIM connector. The unit is in the DTE mode, thus direct connection to the PC (DTE mode) should be made. This connection is given in Table 4.8. Figure 4.4 illustrates the overall signal processing scheme.

Table 4.1 General Specifications – 24 Volt SOLA Supply

Voltage/Current Ratings:

Model Number	**Output**
SLS-05-060-1	+5V/6.0A
SLS-12-034	+12V/3.4A
SLS-24-024	+24V/2.4A

Operating Temperature Range: 0 to +50°C (Derate to 40% at +70°C)

Temperature Coefficient (Typical): +/-0.01%/°C

Stability: Within +/-0.05%(For 24 hours after warm up)

Vibration: Per MIL-STD-810C, Method 514

Shock: Per MIL-STD 810C, Method 516

EMI/RFI: Linear power supplies have inherently low conducted and radiated noise levels. For most system applications, these power supplies will meet the requirements of FCC Class "B" and VCE 0871 for Class "B" equipment without additional noise filtering.

Cooling: Forced air @20 CFM

Input Specifications:

Multi Input (all units): 100/120/220/230/240 VAC selectable +/- 10% except 230 is +15%, -6%

Frequency Range: 47-63Hz. (Typical is 60Hz. Derate output 10% at 50Hz.)

Transient Response Time: 50 *u*SEC at 50% load changes for outputs rated up to 6A
100 *u*SEC at 50% load changes for outputs rated 6 A and over.

Fuse Requirements: Units are *not* fused internally. For safe operation, user must provide input line fuse as per values given in table.

Table 4.1 General Specifications – 24 Volt SOLA Supply(continued)

Output Specifications:

Line Regulation:	0.05% for 10% change
Load Regulation:	0.05% for 50% change
Ripple:	3.0 mV maximum peak to peak
DC Output Adjustment Range:	+/-5% minimum
Overvoltage Protection:	All 5 volt outputs include built-in OVP as standard (setting is 6.2V +/-0/4 V). OVP is optionally available on other outputs.
Remote Sensing:	All units listed have remote sensing capability.
Overload Protection:	125 to 150% foldback current limit
Dielectric Withstand Voltage (Min.):	3750 VAC input/output 1250 VAC input/safety ground 500 VAC output/safety ground

Table 4.2 Condor - GPC55 Series (Multi-voltage Switching Supply)

Ratings

Input: 100-240 VAC, 1.7 A, 47-63Hz

Outputs: 55 Watts Maximum Continuous Power – Total of all Outputs

Model	Watts	Output #1	I_{SC}	Output #2	I_{SC}	Output #3	I_{SC}	Output#4	I_{SC}
GPC55A	55	+5VDC6A	4A	+12VDC3A	4A	+12VDC1A	3A	-12VDC1A	3A

Table 4.3 Condor Connections
(Related to Table 4.2)

J1 AC Input	J2 Multi-Output Models (Reference Table 4.2)	
1) Ground	1) Output 2 (+)	6) Common
2 Neutral	2) Output 2 (+)	7) Common
3) Line	3) Output 1 (+)	8) Output 4 (-)
	4) Output 1 (+)	9)Output 3 (+/-)
	5) Common	

Table 4.4 J1 - VMEbus Signal Descriptions

Signal Mnemonic	Connector And Pin Number	Signal Name and Description
ACFAIL*	IB: 3	AC FAILURE
IACKIN*	1A: 21	INTERRUPT ACKNOWLEDGE IN
IACKOUT*	1A: 22	INTERRUPT ACKNOWLEDGE OUT
AM0-AM5	1A: 23 1B: 16, 17, 18,19 1C: 14	ADDRESS MODIFIER (BITS 0-5)
AS*	1A: 18	ADDRESS STROBE
A01-A23	1A: 24-30 1C: 15-30	ADDRESS bus (bits 1-23)
A24-A31	2B: 4-11	ADDRESS bus (bits 24-31)
BBSY*	1B: 1	BUS BUSY
BCLR*	1B: 2	BUS CLEAR
BERR*	1C: 11	BUS ERROR
BGOIN*-BG3IN*	1B: 4, 6, 8, 10	BUS GRANT (0-3)
BG0OUT*-BG3OUT*	1B: 5, 7, 9, 11	BUS GRANT (0-3)
BR0*-BR3*	1B: 12 - 15	BUS REQUEST (0-3)
DS0*	1A: 13	DATA STROBE 0
DS1*	1A: 12	DATA STROBE 1
DTACK*	1A: 16	DATA TRANSFER ACKNOWLEDGE
D00-D15	1A: 1-8, 1C: 1-8	DATA BUS (bits 0-15)
D16-D31	2B: 14-21 2B: 23-30	DATA BUS (BITS 16-31)
GND	1A: 9, 11, 15, 17, 19 1B: 20, 23 1C: 9 2B: 2, 12, 22, 31	GROUND
IACK*	1A: 20	INTERRUPT ACKNOWLEDGE
IRQ1*-IRQ7*	1B: 24-30	INTERRUPT REQUEST (1-7)
LWORD*	1C: 13	LONGWORD
[RESERVED]	2B: 3	RESERVED
SERCLK	1B: 21	
SERDAT	1B: 22	
SYSCLK	1A: 10	SYSTEM CLOCK
SYSFAIL*	1C: 10	SYSTEM FAIL
SYSRESET*	1C: 12	SYSTEM RESET

Table 4.4 VMEbus Signal Descriptions (continued)

Signal Mnemonic	Connector and Pin Number	Signal Name and Description
WRITE*	1A: 14	WRITE
+5V STDBY	1B: 31	+5 Vdc STANDBY
+5v	1A: 32, 1B: 32, 1C: 32 2B: 1, 13, 32	+5 Vdc STANDBY
+12V	1C: 31	+12 Vdc Power
-12V	1A: 31	-12 Vdc Power
GND	A-10, A 17-A19, A24, A25, A31, A32, C11, C20, C25, C31,C32	GROUND (Logic)
A11	A11	ADDRESS bus (bit 11)
A10	A12	ADDRESS bus (bit 10)
A8	A13	ADDRESS bus (bit 8)
A6	A14	ADDRESS bus (bit 6)
A4	A15	ADDRESS bus (bit 4)
A2	A16	ADDRESS bus (bit 2)
D7	A20	DATA bus (bit 7)
D6	A21	DATA bus (bit 6)
D4	A22	DATA bus (bit 4)
D2	A23	DATA bus (bit 2)
-12V	A26, C26	-12 Vdc Power
(Reserved)	A27, C8-C10, C27	Not connected.
+12V	A28, C28	+12 Vdc Power
+5V	A29, A30, C29, C30	+5 Vdc Power
INT4*	C1	INTERRUPT REQUEST 4
INT3*	C2	INTERRUPT REQUEST 3
INT2*	C3	INTERRUPT REQUEST 2
INT1*	C4	INTERRUPT REQUEST 1
IORES*	C5	INPUT/OUTPUT RESET
XACK*	C6	TRANSFER ACKNOWLEDGE
CLK	C7	CLOCK
A9	C12	ADDRESS bus (bit 9)
A7	C13	ADDRESS bus (bit 7)
A5	C14	ADDRESS bus (5 bit)
A3	C15	ADDRESS bus (bit 3)
A1	C16	ADDRESS bus (bit 1)
A0	C17	ADDRESS bus (bit 0)
STB*	C18	STROBE – An input signal.
WT*	C19	WRITE
D5	C21	DATA bus (bit 5)
D3	C22	DATA bus (bit 3)
D1	C23	DATA bus (bit 1)
D0	C24	DATA bus (bit 0)

Table 4.5 Laser Connections

SIGNAL	PIN
+20	A
+18	B
-18	C
+12	D
Gnd	E
Gnd	F
DATA	G
DATA NOT	H
CLK	I
CLOCK NOT	J

Table 4.6 Accelerometer Connections

SIGNAL	PIN
Right (Left) Accelerometer In	Center
Ground	Shield

Table 4.7 Acoustic Connections - Channels 1-5

SIGNAL	PIN
Acoustic Channel 1	A
Acoustic Channel 2	B
Acoustic Channel 3	C
Acoustic Channel 4	D
Acoustic Channel 5	E
START	F
GND	G

Table 4.8 Acoustic Connections - Channels 6-10

SIGNAL	PIN
Acoustic Channel 6	A
Acoustic Channel 7	B
Acoustic Channel 8	C
Acoustic Channel 9	D
Acoustic Channel 10	E
START	F
GND	G

Table 4.9 PC Serial Connections

SIGNAL	PIN
TxD	2
RcD	3
Gnd	7

4.2 Power Wiring Flow

DC power wiring is indicated in Figures 4.5 and 4.6. Figure 4.5 provides power wiring for the ±24, ±15, +5 and +12 volts. The Power modules 0 and 1 provide the +24 and -24 volts to the laser power module that, via regulators, provide the ±15 volts. As will be seen in Figure 4.6, the laser power module also provides power to the Selcom lasers. The ±15 volts are used to power the two laser interface modules, signal interface module, and the smart A/D module. The 12 volt supply is sent to three places; the two five volt regulars for digital power, the two laser connectors and the 12 volt fan. In the earlier version which used the Condor power supply, the 5 volt requirement was obtained from the Condor switching supply. As discussed above, two separate 5 volt regulators are used, both powered by the 5 amp 12 volt supply. The first is used to power the K processor module. The second 5 volt regulator is used for the two laser interface modules, the signal interface module, and the smart A/D module.

Figure 4.6 illustrates the main laser power (except for the 12 volts) wiring. As indicated in this figure, the laser power module also provides (via regulators) +20 volts, and ±18 volts. There are two independent sets of regulators providing these three voltages. Table 4.3 provides the pin outs for these voltages. The voltages are sent to the Selcom laser through this connector. Also illustrated in the Figure 4.6 is the -24 volt input from the power module1 to the laser power supply module.

Figure 4.7 illustrates the 110 volt AC power wiring. The 110 volt primary is first sent to the on/off power switch. It is then sent to a 5 amp fuse and then directly to the three power supplies. The secondary is sent to the other coil side on the transformers of the power supplies. Figure 4.8 illustrates the system grounds. Ground lug 1 is attached to the 110 volt AC power grounds and the three power supplies. Ground lug 2 is primarily the digital and signal grounds. Both lugs one and two are attached to each other.

4.3 Signal Flow

Signal flow is illustrated in Figures 4.9 and 4.10. Figure 4.9 provides the wiring for the laser and accelerometer signals. As noted, the data and data not signals, along with the clock and clock not signals are sent from the laser 10 pin connector to the laser interface module. As previously discussed, the laser interface module converts this signal to an analog signal which is sent directly to the signal interface module. This module sends the signal on to the A/D converter of the smart A/D module. The accelerometer signals are sent, via the BNC connector to the 400 hertz filter on the signal interface module and then to the A/D module.

Figure 4.10 illustrates the signal wiring for the acoustic channels, start and distance signal and the reset signal. As noted, the acoustic signals are sent directly to the system processor where they are connected directly to the gate input of each timer. The distance signal, after connecting to the signal interface module is sent to the time/distance switch. From this point, either the distance input signal or a separate time signal (with a user programmable period) is selected. The monetary reset switch is used to initialize the processor module.

System Layout - Basic

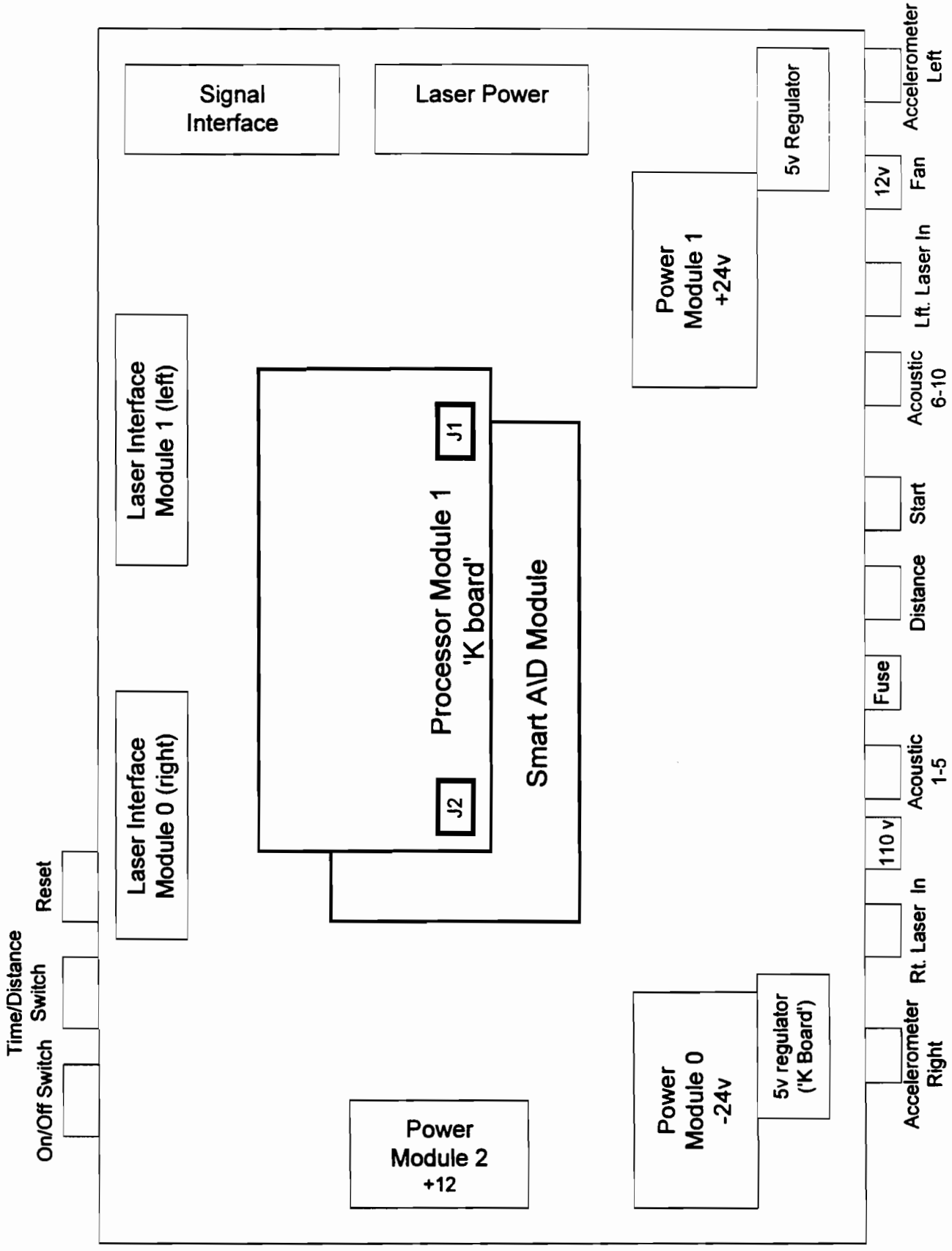


Figure 4.1 Basic System Layout

Profile Measurement System - Front Panel

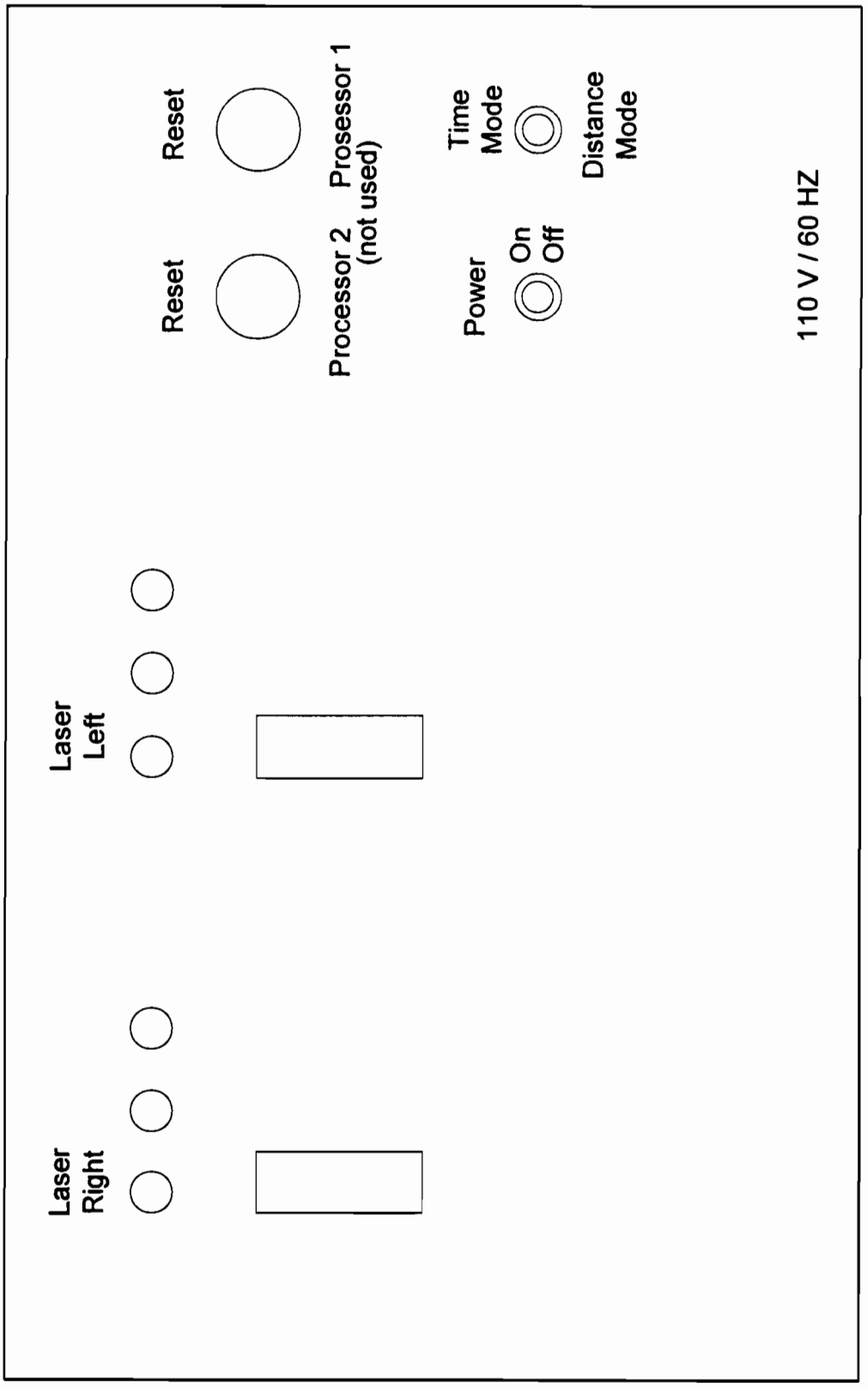


Figure 4.2 Front Panel

Profile Measurement System - Back Panel

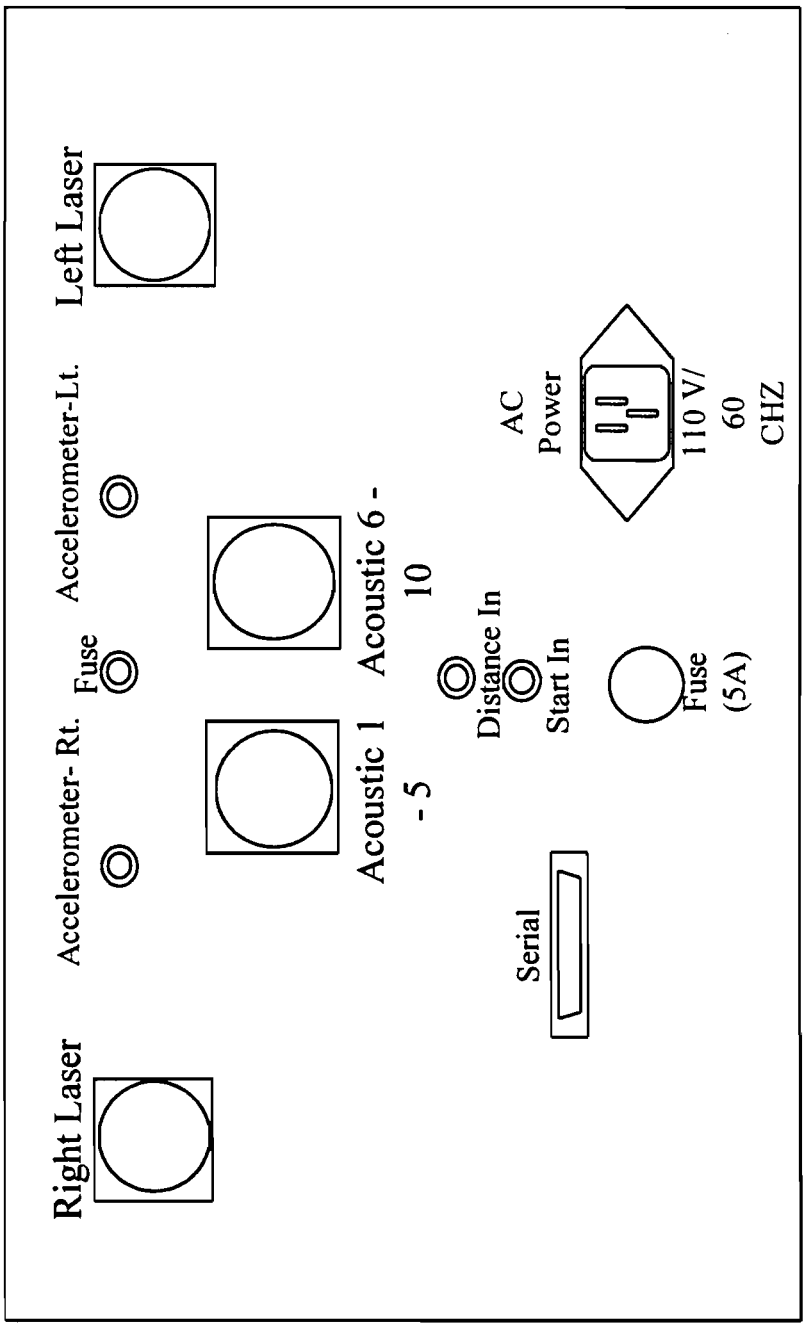


Figure 4.3 Back Panel

System Layout - Wiring 1

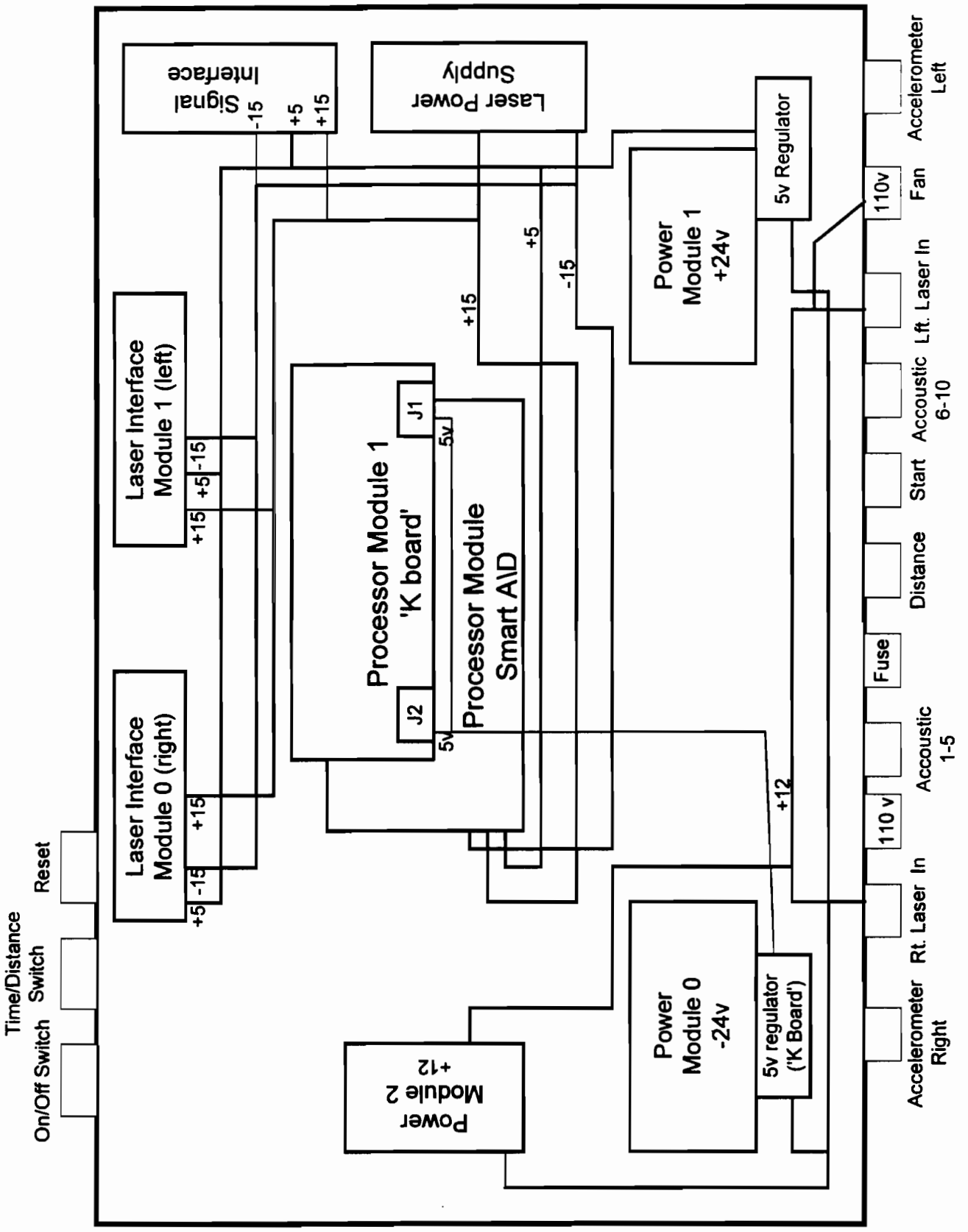


Figure: 4.4 Wiring 1
+5, +12, +15 and -15 Power Connections

System Layout - Wiring 2

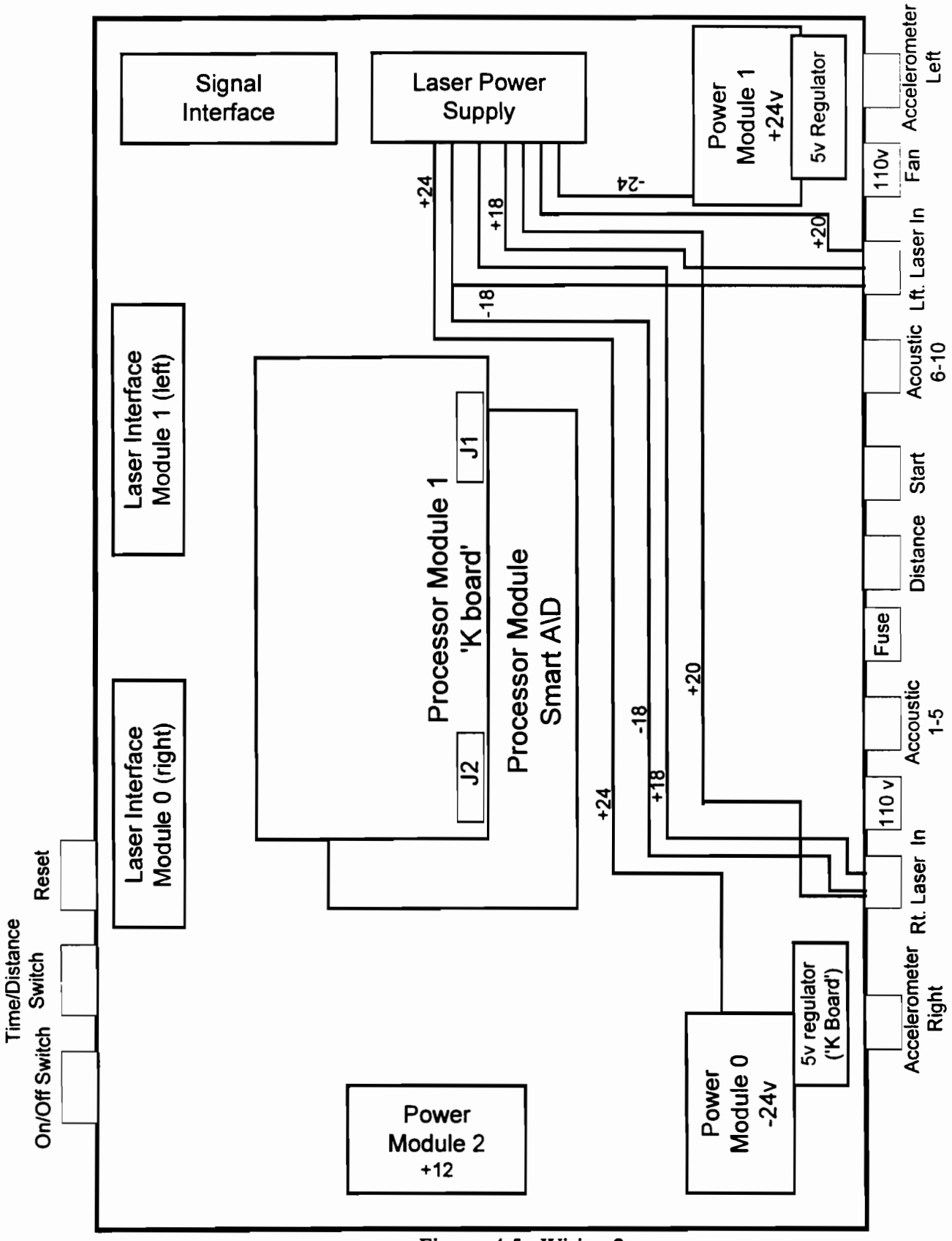


Figure 4.5 Wiring 2
+20, +18, _18, _24, -24 Power Connections

System Layout - Wiring 3

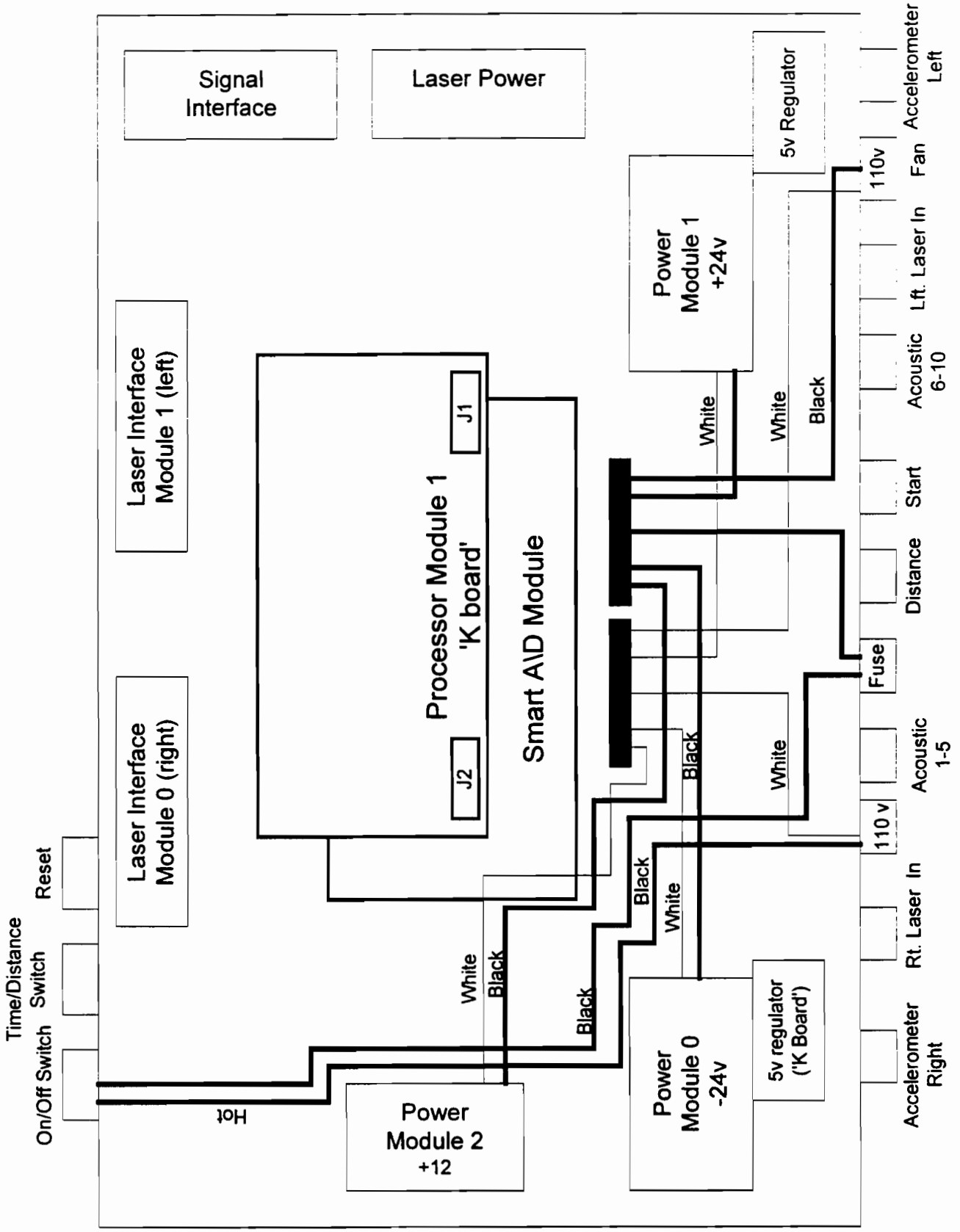


Figure 4.6 Wiring 3
AC Power

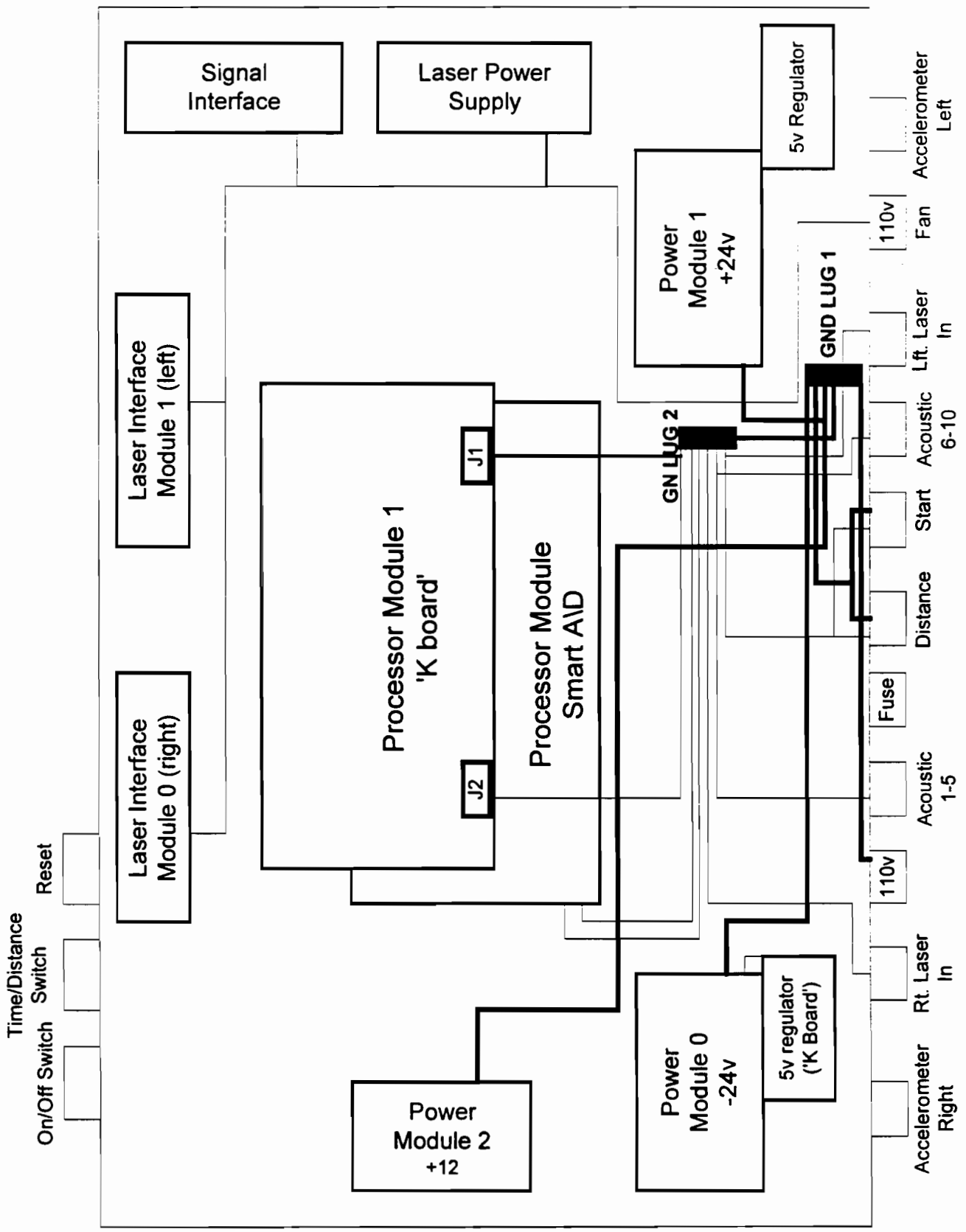


Figure 4.7
Wiring 4 - Grounds

Laser\Accelerometer A/D Interface
System Layout - Signal Wiring 1

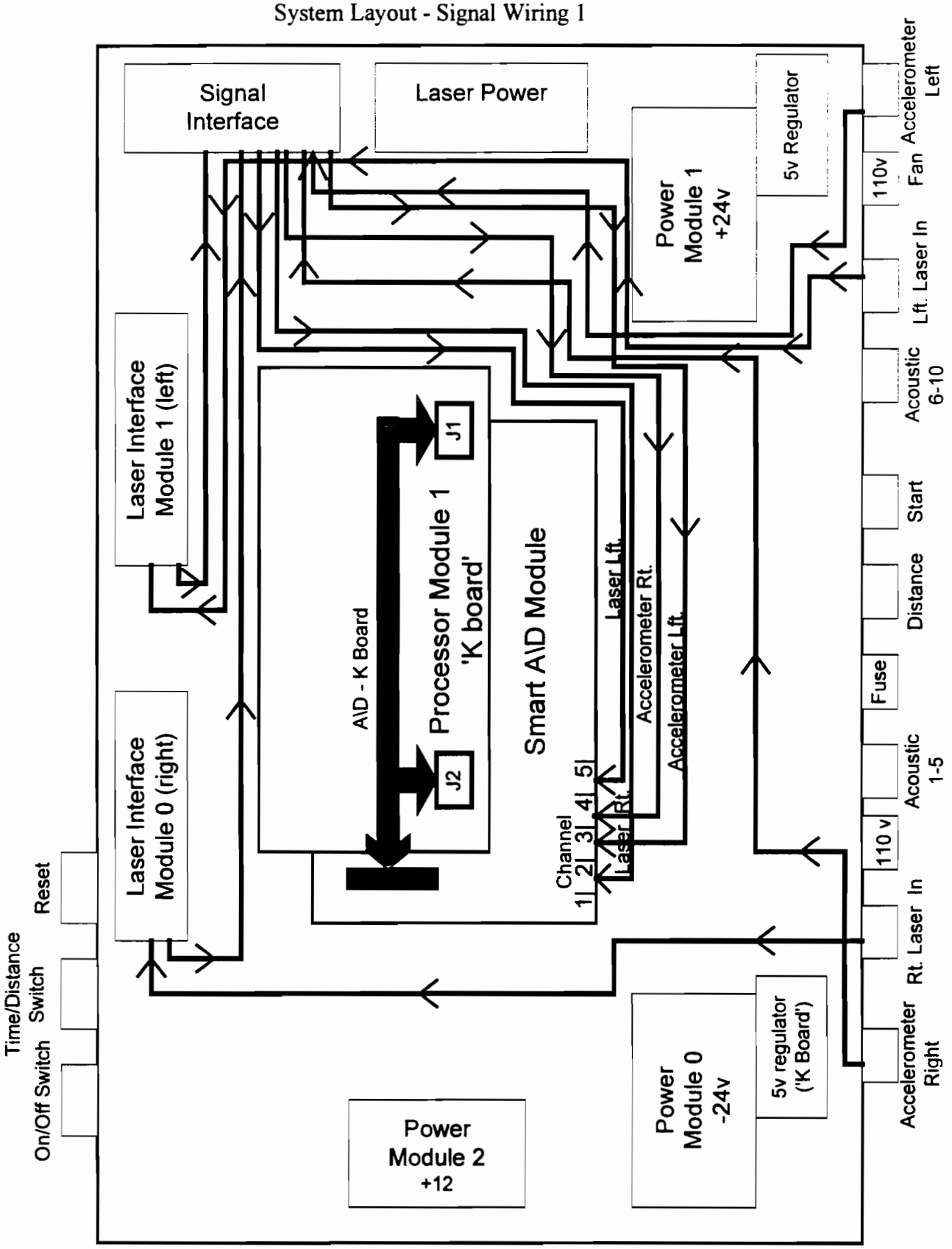


Figure 4.8 Laser/Accelerometer A/D Interface

System Layout - Signal Wiring 2

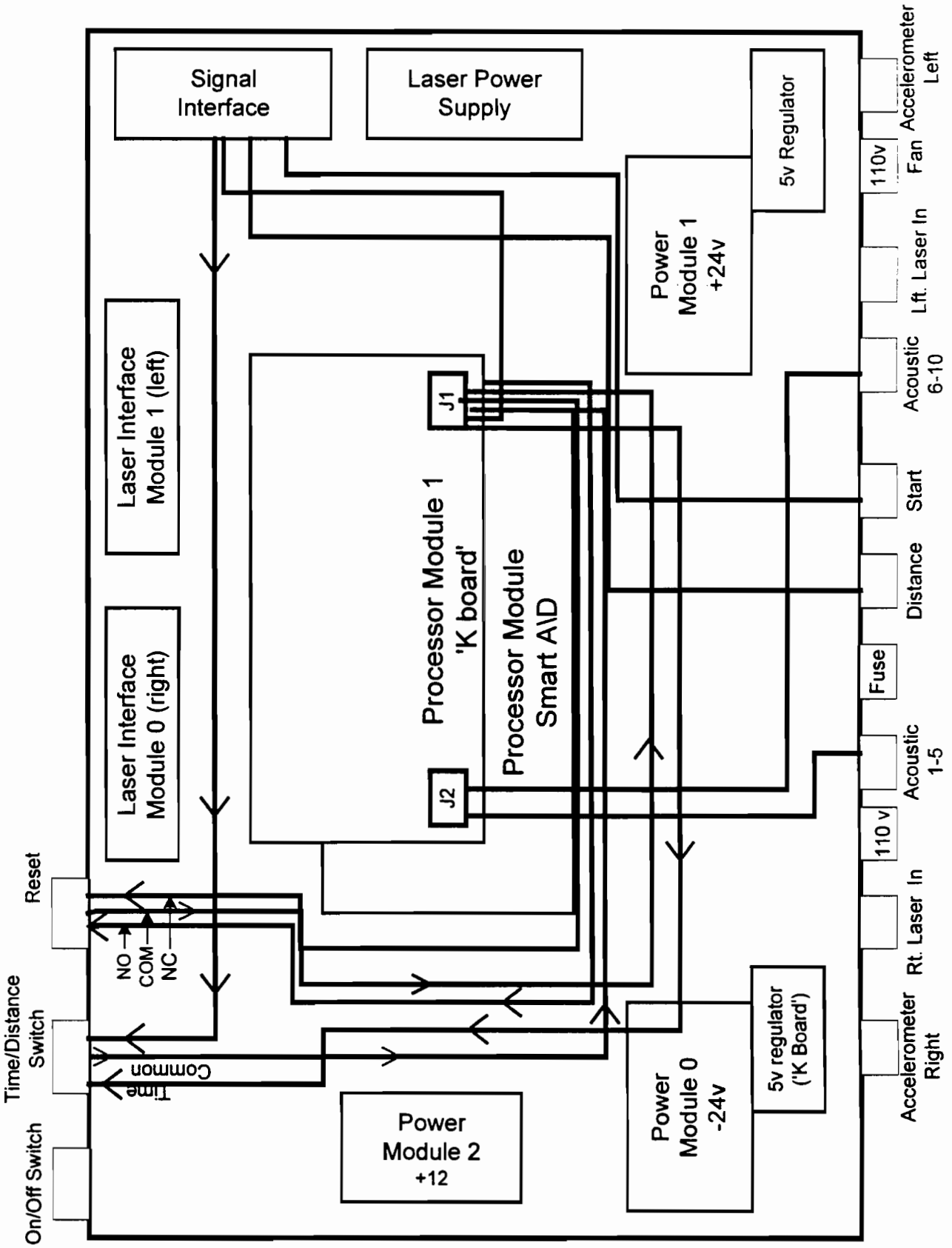


Figure 4.9 Acoustic\Start\Distance\Reset

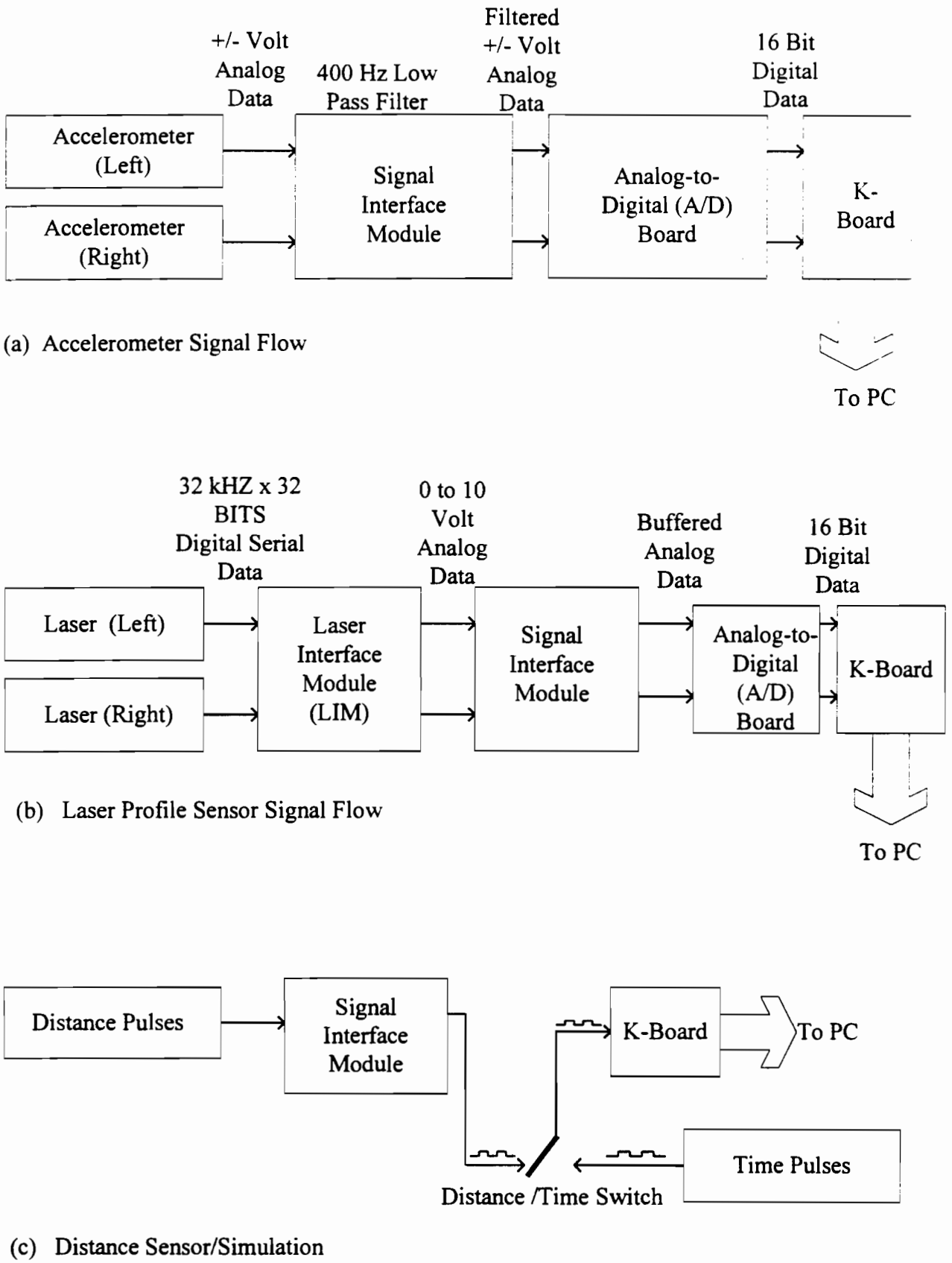


Figure 4.10 Profile Sensors Signal Flow

CHAPTER 5
Laser Power Module

5.0 Overview

This section provides technical information related to the laser power supply module. Also included in this section is a complete description of the printed circuit board (PCB) design used for implementing this module. Although this and the next four chapters discuss design modules which have been implemented as a PCB, only this chapter will include the complete PCB design. The designs of the other boards are available to TxDOT, and are also kept at one of the local PCB board construction facilities.

The laser power supply board is designed to supply regulated power to two (2) Selcom optocator units. Each optocator requires power of various DC amplitudes. The laser power supply board utilizes multiple voltage regulators to produce the differing voltages required by the Selcom optocators. The board also provides +/- 15 volts for the smart A/D module, which is described in a later chapter. Design concepts are provided in the schematic at the end of the chapter.

5.1 Power Considerations

The laser power supply derives its power from an external +/-24VDC power supply. The laser power supply generates the output voltages defined in Table 5-1. A separate voltage regulator is utilized to generate each output.

Output Voltage	Max Current (Amps DC)	Number of Outputs
+ 20 VDC	0.25	2
+18 VDC	0.13	2
- 18 VDC	0.13	2
+ 15 VDC	0.11	1
- 15 VDC	0.11	1

Table 5.1 Output Voltages

5.1.1 External Connections

Input and output power connections are made at terminal blocks J1 – J5.

5.1.2 Input Power Connections

An external +/-24VDC power supply is connected to the laser power supply board at terminal block J1. Make connections as marked on the board.
External Power Connections: The laser power supply board supplies power to two (2) independent Selcom optocators. Each optocator requires power of the following magnitudes: 20VDC, +/-18VDC and 12-15VDC. The 20VDC, and +/-18VDC supplies are made available to each laser at terminal blocks J3 and J4. A common grounding block is provided at terminal block J2. Routing of the 15VDC supply depends on the configuration of the overall system. In some systems, an external 12VDC supply is used to power the 12-15VDC supply. In this configuration, the laser power supply board's +/-15VDC outputs are utilized to power other analog boards. If an external 12VDC supply is not available, the 15VDC supply may be used. Be careful not to exceed the current limitations defined in Table 5.1. Figure 5.1 provides a schematic of laser power module. Figures 5.2 through 5.6 illustrate the PC Board design criteria. Figure 5.7 provides a plot of the board layout and signal interface.

5.1.3 Laser Power Supply Board Parts List

Quantity	Part #	Description	Reference Designator(s)
2	MC78M20CT	+20VDC Voltage Regulator	U1, U2
2	MC7818CT	+18VDC Voltage Regulator	U3, U4
2	MC7918CT	-18VDC Voltage Regulator	U5, U6
1	MC7815CT	+15VDC Voltage Regulator	U7
1	MC7915CT	-15VDC Voltage Regulator	U8
8		Cap. 0.33 µF Tantalum	C1 – C8
8		Cap. 10 µF Tantalum	C9 – C16
3		3 Position Terminal Block	J1, J3, and J4
1		7 Position Terminal Block	J2
1		2 Position Terminal Block	J5

Table 5.2 Parts List Laser Supply Board

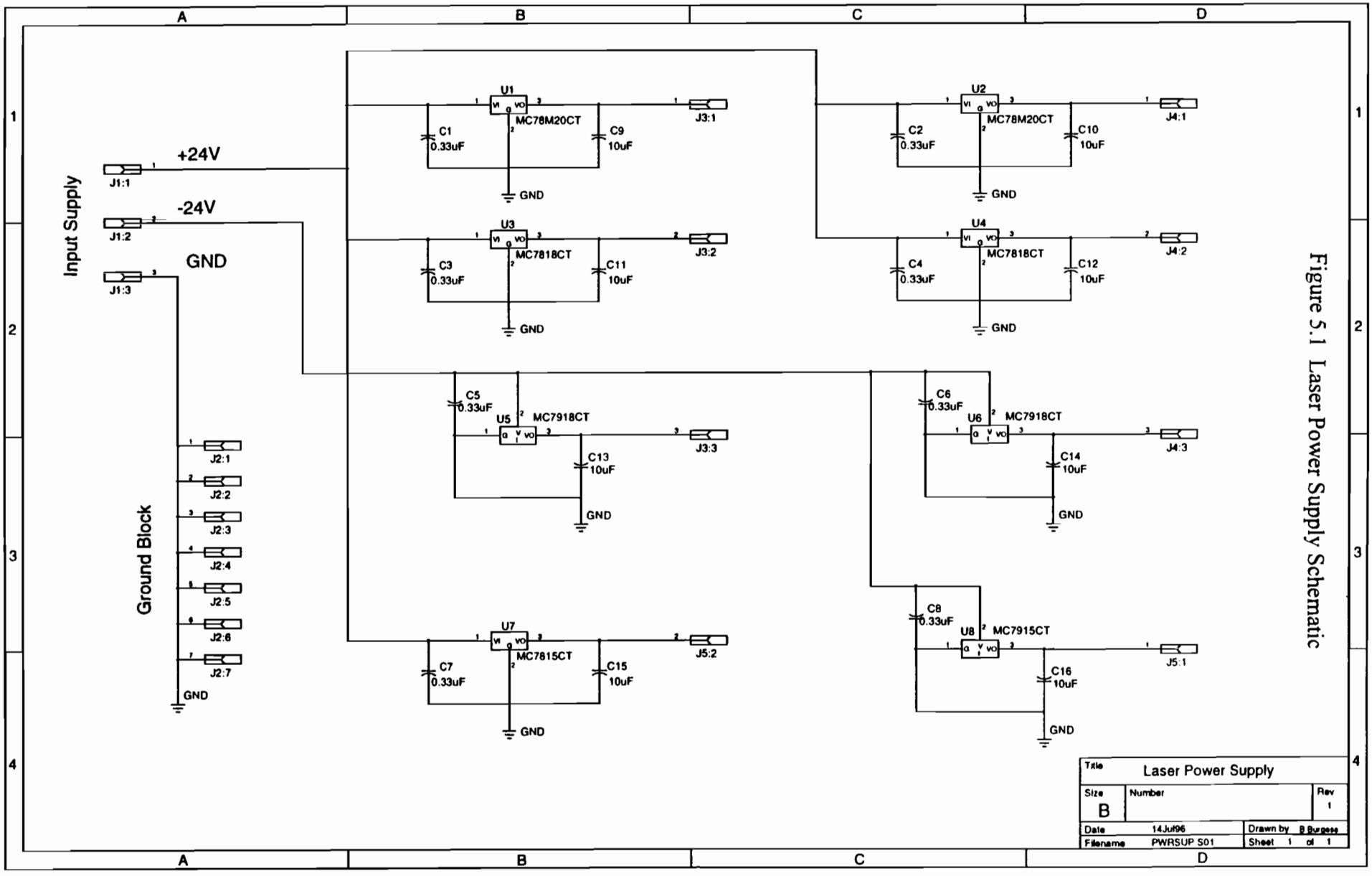
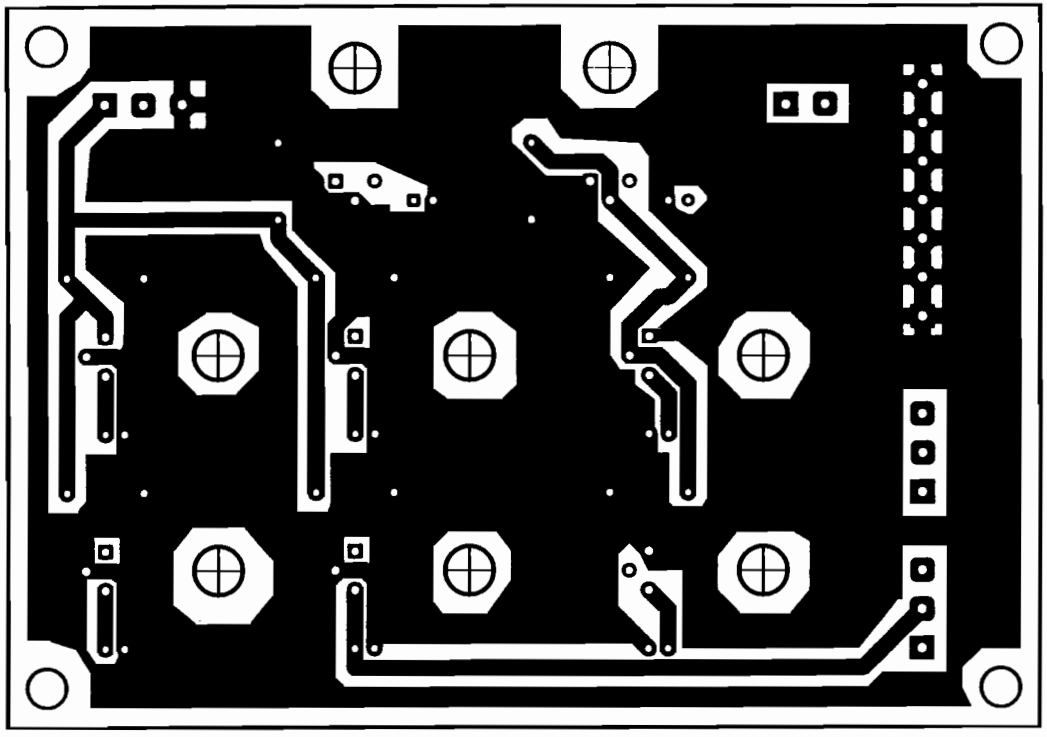


Figure 5.1 Laser Power Supply Schematic



Title	Power Supply PCB		
Size A	Number Bottom		Rev 1
Date	14Jul96	Drawn by	B.Burgess
Filename	PWRSUP.PCB	Sheet 1 of 1	

Figure 5.2 Power Supply PC Board Bottom View

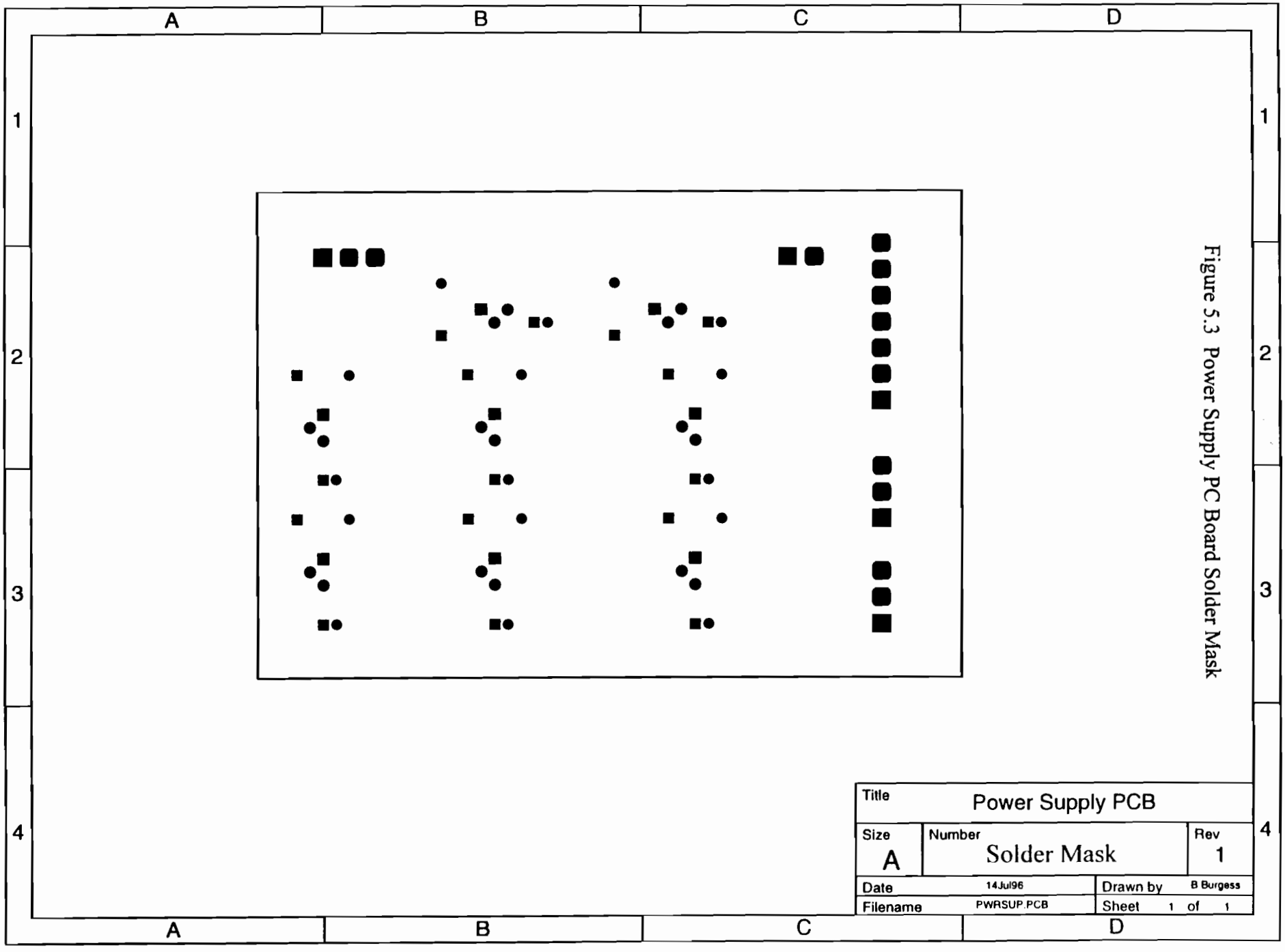
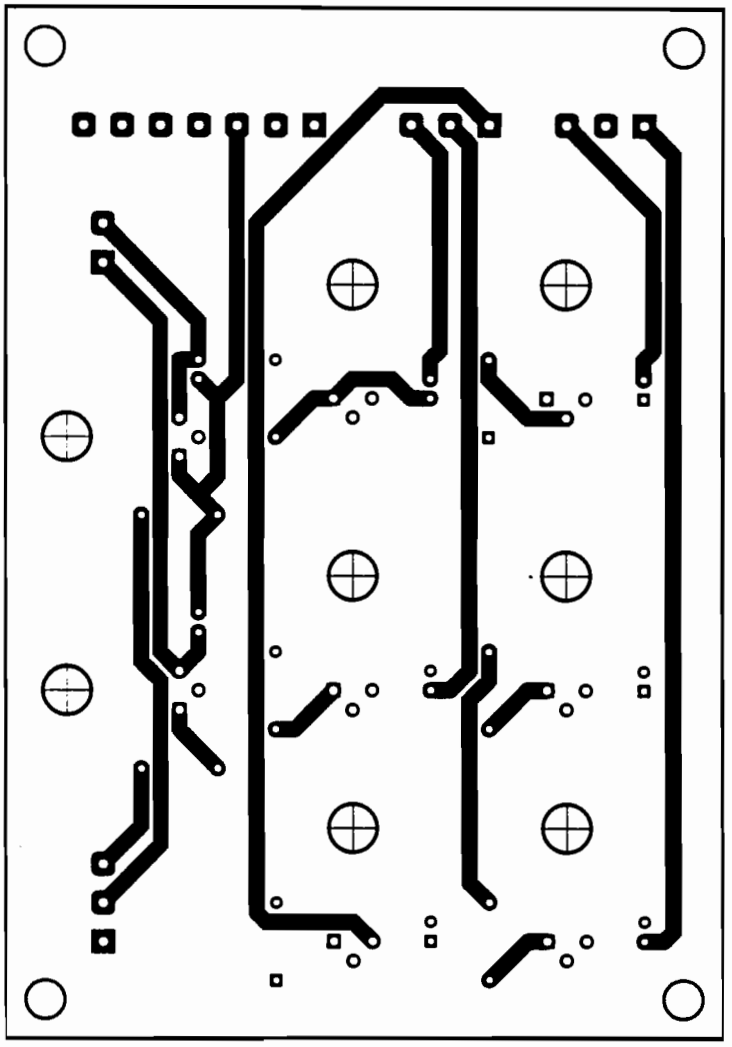


Figure 5.3 Power Supply PC Board Solder Mask

Figure 5.4 Power Supply PC Board Top View



Title	Power Supply PCB		
Size: A	Number: Top	Rev: 1	
Date: 14Jul96		Drawn by: B.Burgess	
Filename: PWRSUP.PCB		Sheet: 1 of 1	

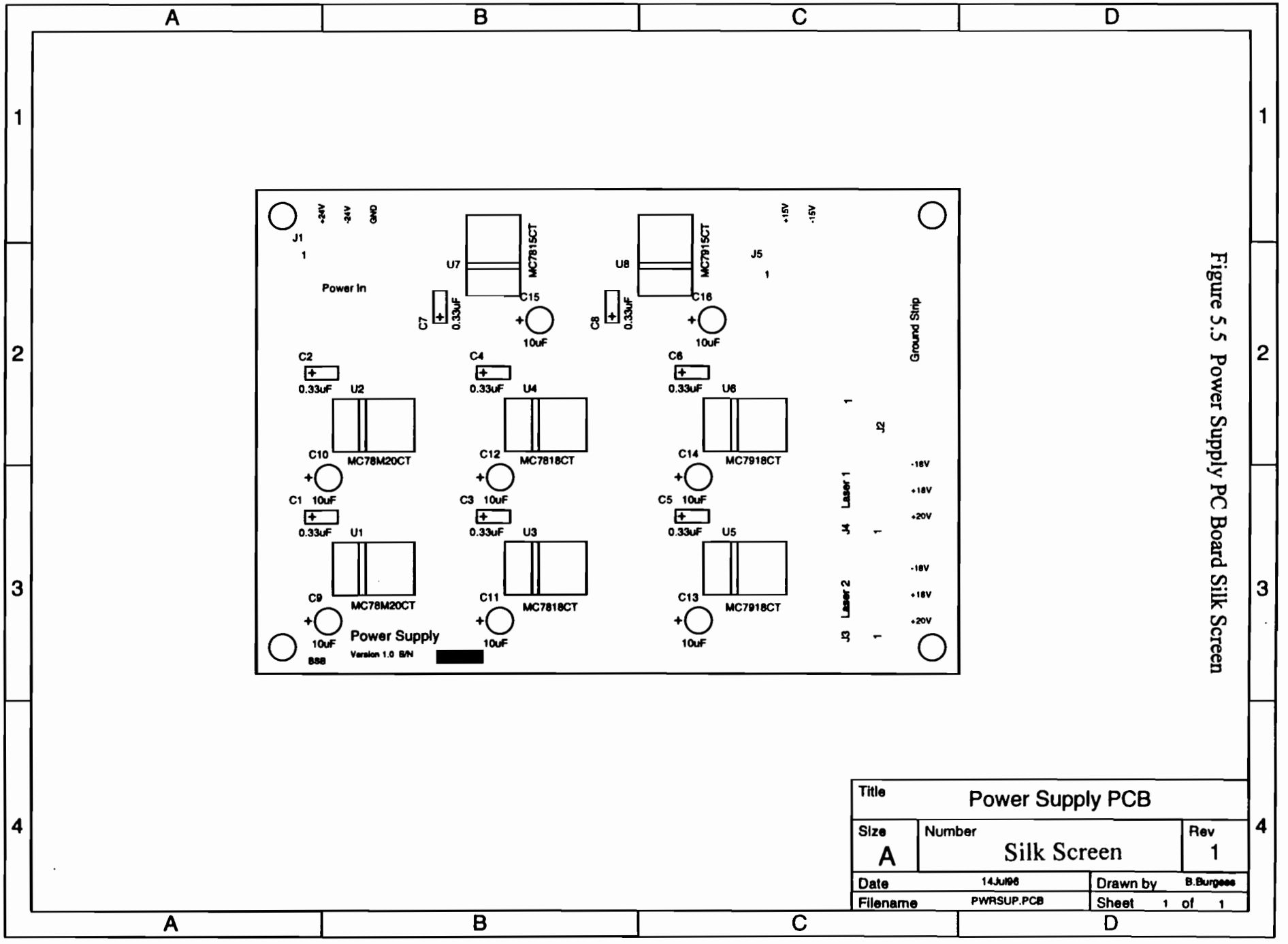
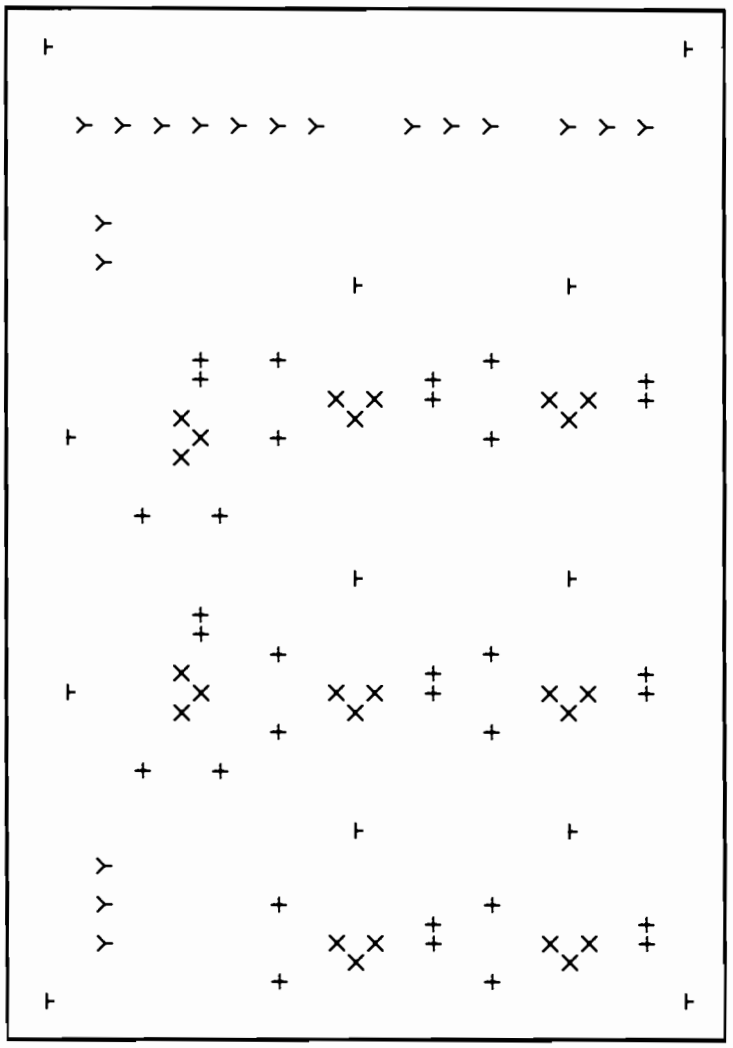


Figure 5.5 Power Supply PC Board Silk Screen

Figure 5.6 Power Supply PC Board Drill Schematic



Title	Power Supply PCB		Rev 1
Size A	Number	Drill	
Date	14Jun98	Drawn by	B.Burgess
Filename	PWRSUP.PCB	Sheet	1 of 1

(in mills)
Y = 50
X = 46
+ = 38
T = 150

LASER POWER BOARD

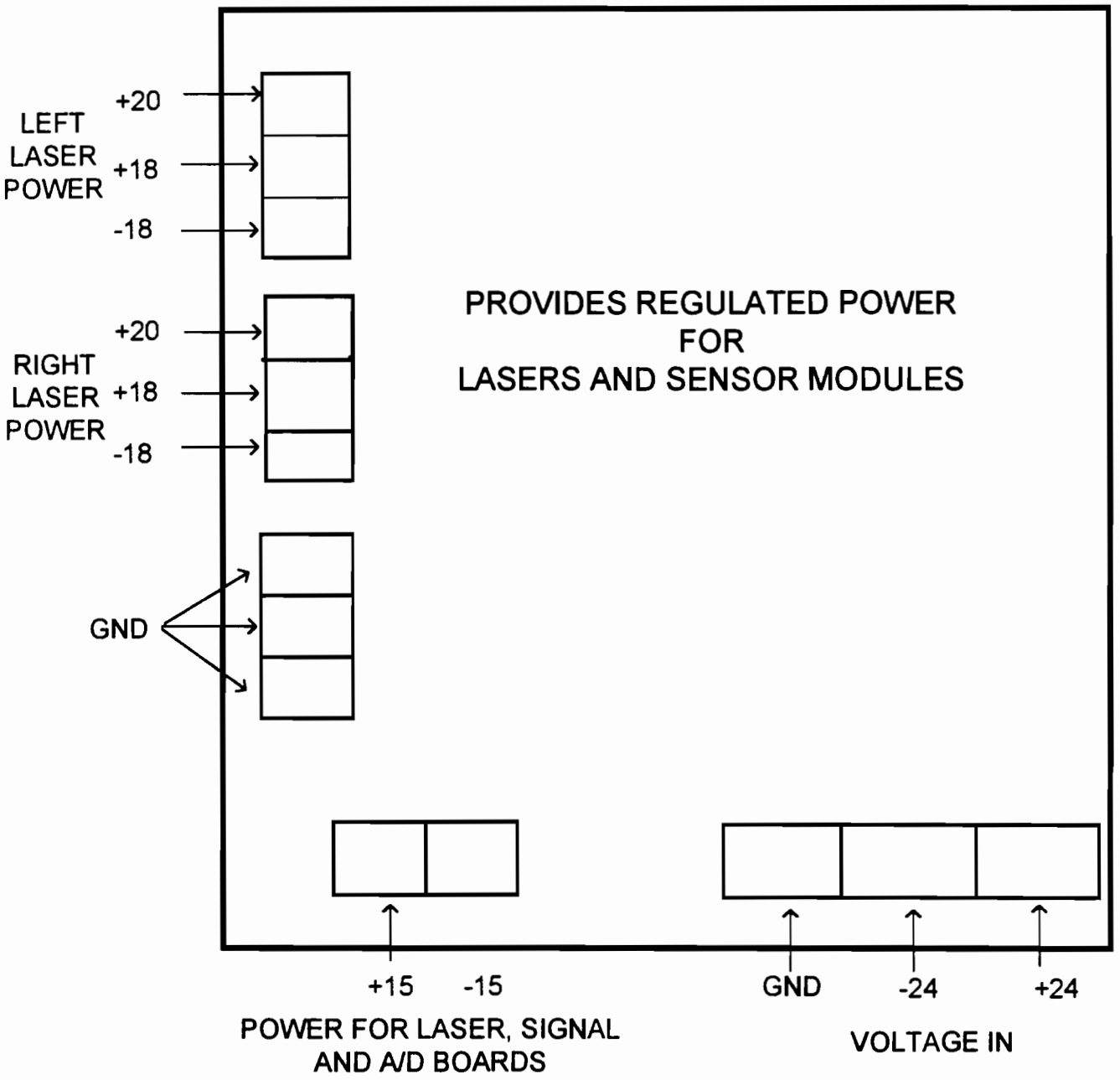


Figure 5.7 Laser Power Board

CHAPTER 6
Signal Interface Board

6.1 Overview

This section details the design of and serves as a reference for the Signal Interface Board (SIB), hereafter referred to as the SIB. The SIB performs signal conditioning for specified K processor board inputs. The SIB provides two (2) low pass filters, two (2) single ended analog buffers, and optical isolation for two (2) discrete signals. One (1) of the discrete signal interfaces is equipped with a "divide down" capability. This function is user selectable and intended to allow high frequency distance pulses to be scaled into a lower frequency range. A light emitting diode (LED) provides visual representation of the logic level on the remaining discrete signal.

6.2 Detailed Design\SIB Signals

This section provides the detailed design of the SIB. The SIB interfaces to four (4) types of signals. The following paragraphs identify the different signal types and the corresponding conditioning performed for each.

6.2.1 Left and Right Accelerometers

The SIB interfaces with the left and right accelerometers. Each accelerometer signal is passed through a low pass filter (400 Hz). The filter outputs are accessible on the J3 connector. These signals may then be routed to an A/D converter for processing by the K processor board. The low pass filters remove any high frequency components that may be introduced as a result of vehicle vibration and electromagnetic interference (EMI).

6.2.2 Start Signal

The start signal indicates the beginning of a section of road. A device generates the signal as the unit passes over a white stripe. This is accomplished by emitting a beam of light and detecting its reflection off of a bright object (the pavement itself does not reflect the light). The start signal is electrically grounded during the presence of the stripe, otherwise it is electrically open. The signal is optically isolated and buffered. The output is available on connector J4 such that it can be routed to the K processor board. The SIB is equipped with an LED such that the current state of the start signal can be easily determined. The LED illuminates when the sensor detects reflected light, as when it passes over the white strip. The LED allows for quick alignment and operational checkout of the sensor.

6.2.3 Distance Signal

The distance signal is generated by a sensor that generates pulses at a frequency proportional to the speed of the vehicle. By counting the number of pulses during a period of

time, the distance traveled may be determined. This signal is optically isolated and buffered. Additionally, jumper block JP1 allows the user to "divide down" the pulses. The "divide down" function allows high frequency ranges to be mapped into a lower range thus generating fewer processor interruptions. This function also allows the outputs of different sensors that produce different frequencies for the same speed, to be mapped into the same range.

6.3 Hardware Installation

This section provides the installation instruction for the SIB. Installation consists of configuring JP1 and connecting external wiring. The following paragraphs detail these steps.

Jumper Configuration: Jumper block JP1 must be configured for the required scaling of the distance signal. Table 6-1 defines the configuration settings for JP1.

Jumper Pins	Corresponding Output Frequency
1 – 2	Fout = Fin
3 – 4	Fout = Fin / 2
5 – 6	Fout = Fin / 4
7 – 8	Fout = Fin / 8
9 – 10	Fout = Fin /16
11 – 12	Fout = Fin / 32
13 – 14	Fout = Fin / 64
15 – 16	Fout = Fin / 128

Table 6.1 Jumper Block

6.3.1 Configuration External Connections

Power, input signals, and output signals are all connected to the SIB via terminal blocks J1, J2, J3, J4, and J5. The following paragraphs define the connections for each signal.

Power to the SIB is applied at terminal block J1. The board requires +5V DC and +/- 12V DC supplies. The grounds of +5V and +/- 15V supplies are assumed to be common. This ground is connected to the GND connector of J1. If the supplies do not have a common ground, they must be connected together either externally, or at the GND input of J1. When connecting supply grounds together, be aware of the employed grounding system. Use a scheme that prevents ground loops.

6.3.2 Accelerometer Input and Output Signal Connections

The accelerometer inputs are connected to terminal J2. The left and right accelerometer inputs connect to terminals LAI and RAI respectively. The filtered accelerometer outputs are connected terminal block J3. The left and right outputs connect to terminals LAO and RAO respectively.

6.3.3 Laser Input and Output Signal Connections

The laser inputs are connected to terminal J2. The left and right laser inputs connect to terminals LLI and RLI respectively. The laser outputs are connected to terminal block J3. The left and right laser outputs connect to terminals LLO and RLO respectively: Distance Signal Input and Output Connections

The distance input and output signals are connected to the SIB at terminal block J4. The two terminals are labeled "Dist", with an "I" and "O" identifying the input and output terminals respectively.

6.3.4 Start Signal Input and Output Connection

The start input and output signals are connected to the SIB at terminal block J4. The two terminals are labeled "Start", with an "I" and "O" identifying the input and output terminals respectively.

6.4 Signal Interface Board Layout

Figure 6.1 and Figure 6.2 provides a plot of the board layout, and a schematic of the board.

SIGNAL INTERFACE BOARD

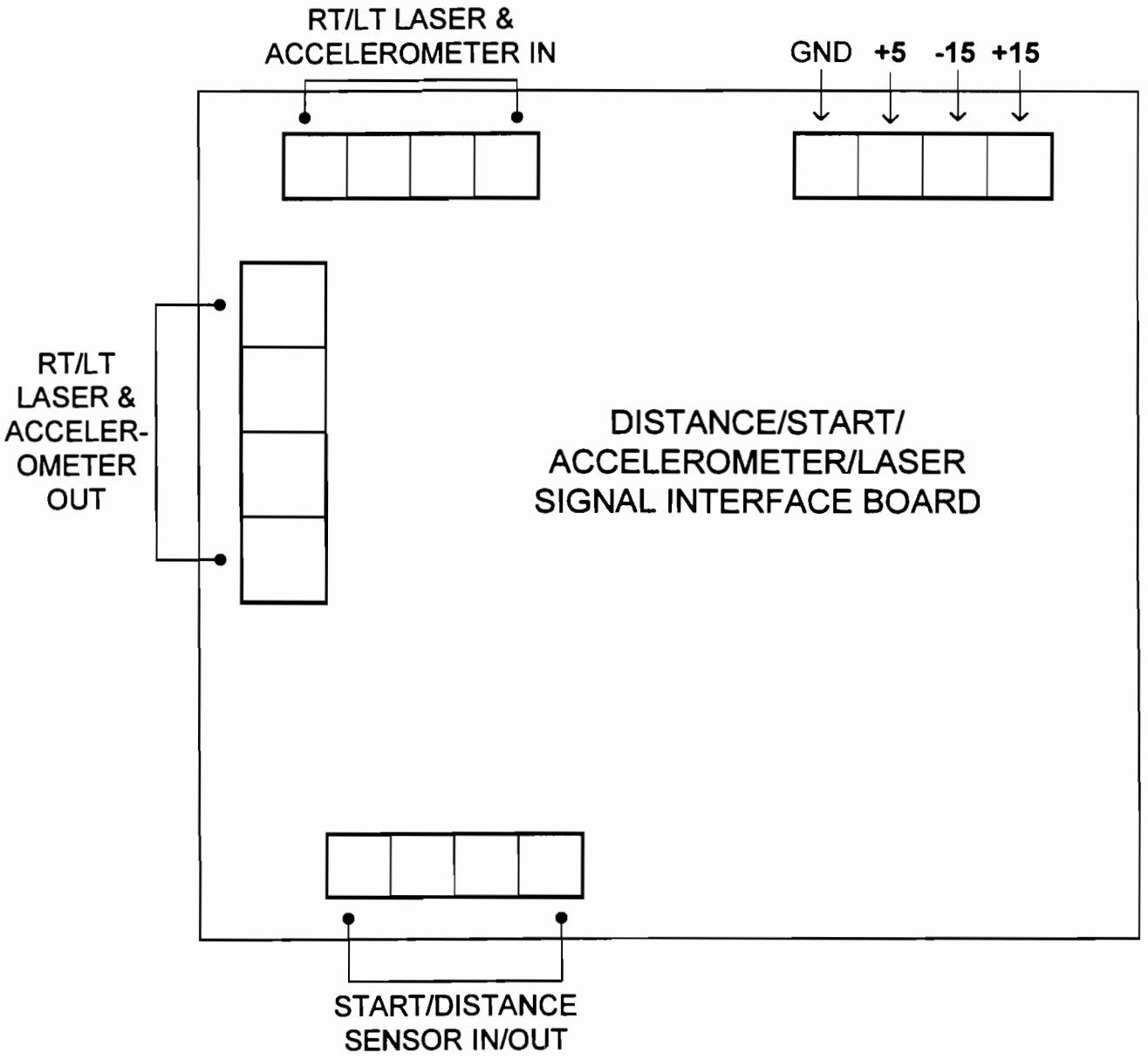


Figure 6.1 Signal Interface Board

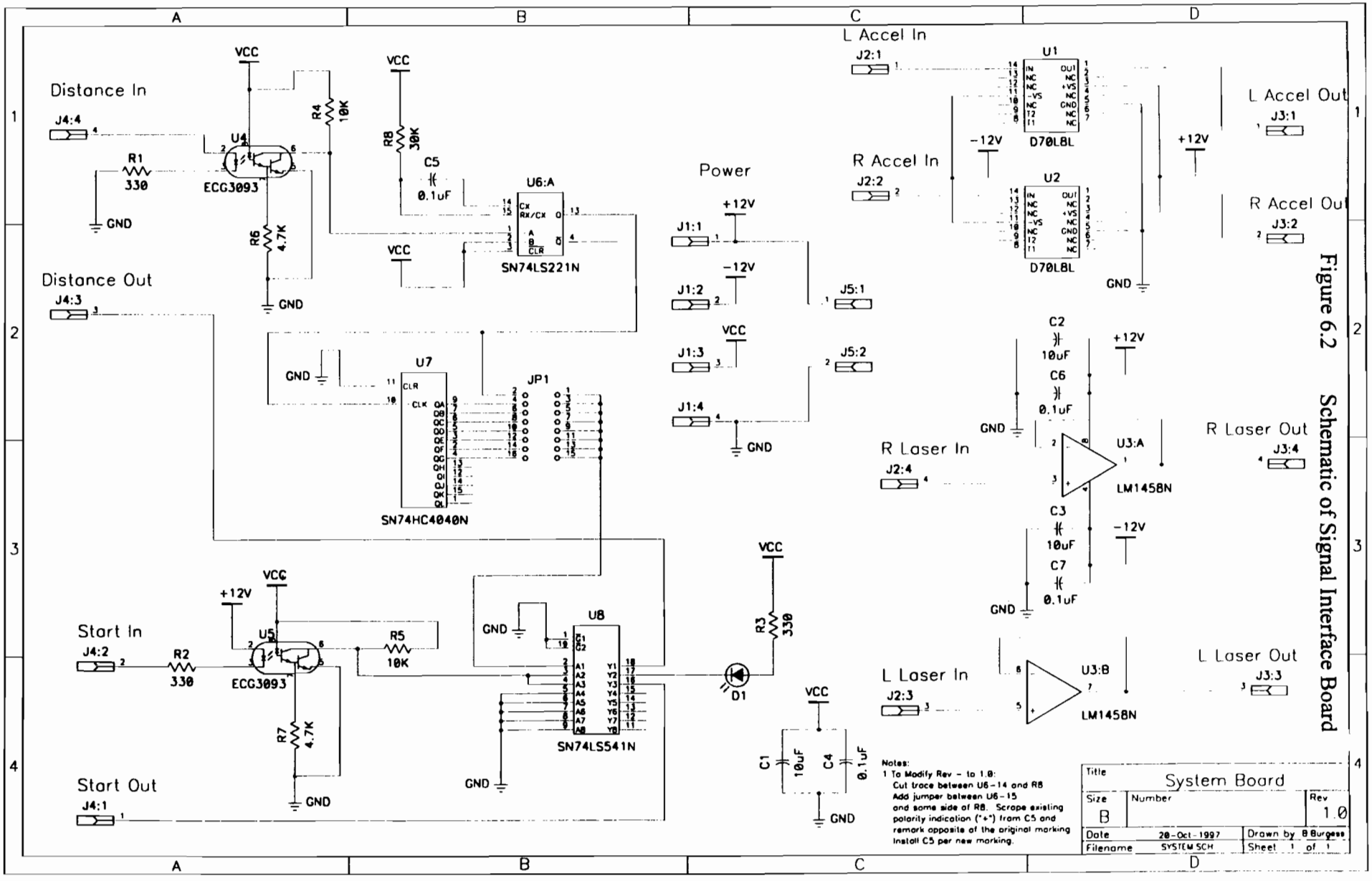


Figure 6.2 Schematic of Signal Interface Board

Table 6.2 SIB Parts List

Quantity	Part #	Description	Reference Designator(s)
2	D70L8L	400 Hz Filter	U1, U2
1	LM1458	Operational Amplifier	U3
2	ECG3093	Opto-Isolator	U4, U5
1	SN74LS221N		U6
1	SN74HC4040N	Binary Counter	U7
1	SN74LS541N	Buffer	U8
1		Red LED	D1
3		Res. 330 Ω, 5%	R1, R2, R3
2		Res. 10 KΩ, 5%	R4, R5
2		Res. 4.7 KΩ, 5%	R6, R7
1		Res. 30 KΩ, 5%	R8
3		Cap. 10 µF Tantalum	C1, C2, C3
4		Cap. 0.1 µF Tantalum	C4, C5, C6, C7
3		8 Position Jumper lock	JP1
4		Four Terminal Block	J1, J2, J3, J4
1		Two Terminal Block	J5

Assembly Notes:

To convert revision – board to revision 1.0:

1. Cut trace between U6-14 and R8.
2. Add jumper between U6-15 and same side of R8.
3. Reverse the polarity marking of C5 by scraping off the original and remarking.
4. Install C5 per the new marking.

CHAPTER 7
LASER INTERFACE MODULE

7.0 Overview

This section provides operating instructions and general information for the use of the Laser Interface Module (LIM). The purpose of the Laser Interface Module is to interface with a Selcom Optocator Laser. A Selcom Optocator is used to measure distances with a high degree of accuracy. The Optocator outputs the distance that it is measuring in a serial format. The Laser Interface Module reads this data, performs some averaging, and then sends the averaged values to a digital to analog converter. The output of the Laser Interface Module is a DC voltage (0 – 10 volts) that corresponds to the distance being measure by the Optocator.

7.1. Laser Interface Module Layout

This board uses the Motorola M68HC11. Following is a list of major components and their location on the LIM:

- XC68HC711E9FS – This is an inexpensive 16-bit micro controller. It is packaged in a 52 pin plcc, and is located in the center left portion of the board.

- MACH130 – This is a large scale erasable/programmable logic device. The MACH130 is packaged in an 84 pin plcc, and is located to the right of the 68HC11.

- SPDAC87 – This is the digital to analog converter. It is located below the MACH 130.

- P1 – This connector supplies the 5 volts to the board. It is located on the upper right corner of the LIM.

- J1 – This connector provided the power to the A/D converter, and is also the interface to the Selcom Optocator. This connector is located on the lower right corner of the LIM.

- Bargraph – The bargraph provides a rough visual representation of the distance that is currently being measured by the Optocator. The larger the distance being measured, the larger the bargraph will read. The bargraph is located to the right of the MACH130.

- L1 – This green LED is lit when 5 volts is applied to LIM.

- L2 – This red LED is lit when power is applied to the LIM and not the Optocator. It is also lit if the LIM is not connected to an Optocator.

- L3 – This red LED is lit when the data being sent to the LIM from the Optocator is invalid.

- L1 –L3 located in the upper-right corner above the bargraph display gives the connections.

The Laser Interface Module connections are provided in Table 7.1.

Signal Name	Connector Pin
Ground	P1-1
+5 Volts	P1-2
+12 Volts	J1-1
-12 Volts	J1-2
Vout	J1-3
CLOCKN	J1-5
CLOCK	J1-6
DATAn	J1-7
DATA	J1-8

Table 7.1 Laser

7.2. Functionality

The Laser Interface Module uses a MACH130 to read the serial data from the Selcom Optocator. The MACH determines if the data is valid, and sends this data in a parallel form to the HC11 which performs the averaging. If the data is not valid, then the MACH130 lights LED L3, and does not send the data to the HC11. After the HC11 averages the data, it sends the averaged value to a digital to analog converter. The output of the digital to analog converter is a range from 0 to 10 volts.

Sixteen (16) bit data words are sent to the LIM from the Optocator in a serial format. This is accomplished with two (2) twisted pair lines. One (1) twisted pair carries the data, while the other carries a clock. Every time the clock line shifts from a logic low to a logic high, a new bit is ready to be received on the data lines. The clock has a frequency of one (1) Megahertz. After all sixteen bits are sent from the Optocator, the clock signal remains at a logic low for sixteen cycles. This allows external logic to determine the location between serial words.

The serial data sent from the Optocator has the following format. It is the sixteen (16) bit word, with the most significant bit sent first. This word is divided, with the twelve (12) most significant bits representing the current distance being measured by the Optocator. The three (3) least significant bits are 'invalid' bits, and the distance bits have no meaning.

The MACH130 acts as a simple shift register in reading the data from the Optocator. Every time the clock transitions from a logic low to a logic high, the MACH shifts in another bit from the data line. The MACH utilizes two internal counters. The first counter is used to determine when the shift register is full. The other counter is used to determine if an Optocator is connected. The clock output of the Optocator clears this counter. If the counter becomes full without being cleared, then the Optocator is either not connected or not powered.

The transfer mechanism between the 68HC11 and the MACH130 consists of the twelve (12) data lines and a control line. The control line (README), is active low. Every time README transitions from a logic high to a logic low, new data valid on the data lines. The HC11 polls README, and when it is a logic low reads the new data from the MACH130.

Table 7.2 provides the Pal equation, and HC11 program used for the board. Figure 7.1 is the board schematic while Figure 7.2 provides a plot of the board.

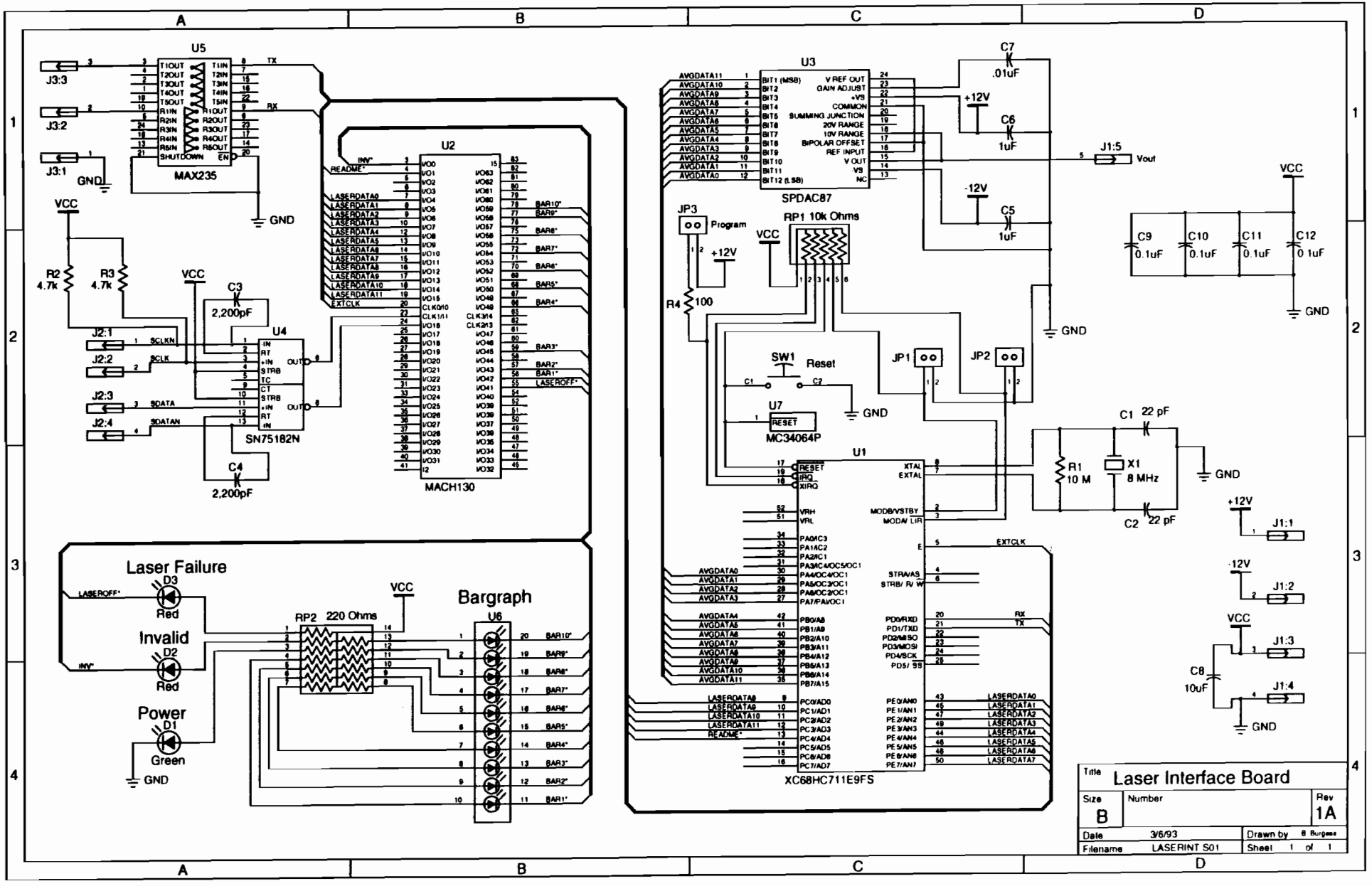


Figure 7.1 Laser Interface Board Schematic

LASER INTERFACE BOARD

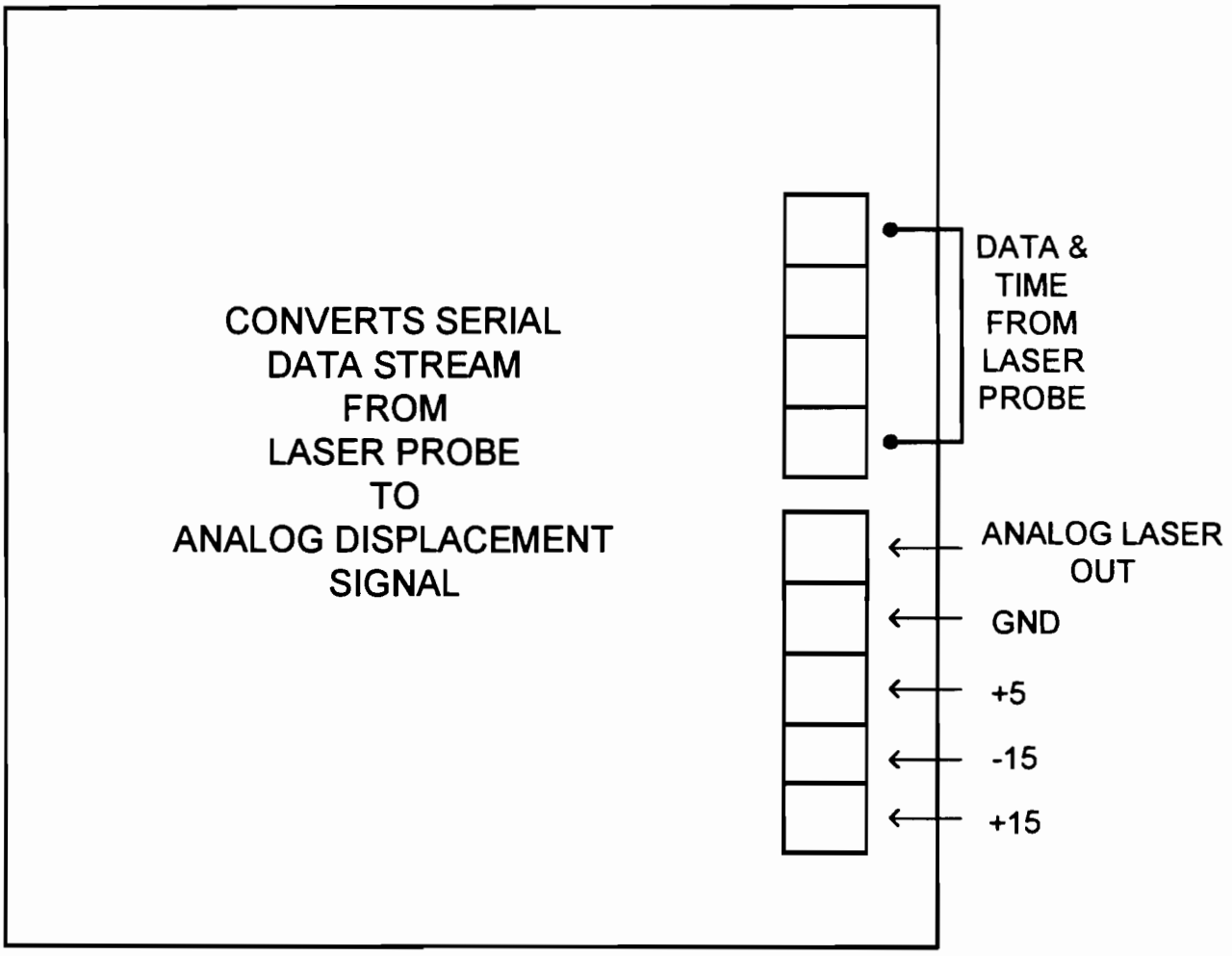


Figure 7.2 Laser Interface Board

Table 7.2 Laser Interface Module PALASM Design Description and HC11 Program

PALASM Design Description

```
   This section contains the listing of the program used in the MACH130.
;---------------------------------- Declaration Segment ------------
TITLE    Laser Interface Module
PATTERN  1-a
REVISION 1-a
AUTHOR   Brian Burgess
COMPANY  UTA
DATE     10/23/92

CHIP  laser  MACH130

;---------------------------------- PIN Declarations ---------------
PIN  20          EXTCLOCK                          COMBINATORIAL ; INPUT
PIN  23          SCLOCK                            COMBINATORIAL ; INPUT
PIN  24          SDATA                             COMBINATORIAL ; INPUT
NODE 18..33      SHIFTREG[0..15]                   REGISTERED    ; INT
PIN  7..10,12..19 DATAOUT[0..11]                   COMBINATORIAL ; OUTPUT
NODE 34..36      CTR1[0..2]                        REGISTERED    ; INT
NODE 37..42      CTR2[0..5]                        REGISTERED    ; INT
NODE 43          CTR2[6]    PAIR LASEROFF          COMBINATORIAL ; OUTPUT
PIN  55          /LASEROFF                         COMBINATORIAL ; OUTPUT
PIN  56,57,59    /LED[1..3]                        COMBINATORIAL ; OUTPUT
PIN  66,68,70    /LED[4..6]                        COMBINATORIAL ; OUTPUT
PIN  72,75,77    /LED[7..9]                        COMBINATORIAL ; OUTPUT
PIN  78          /LED[10]                          COMBINATORIAL ; OUTPUT
PIN  3           /INV                              COMBINATORIAL ; OUTPUT
PIN  4           /README                           COMBINATORIAL ; OUTPUT
GROUP MACH_SEG_B SHIFTREG[0..15]
;---------------------------------- Boolean Equation Segment ------
EQUATIONS
SHIFTREG[0] = SDATA
SHIFTREG[1..15] = SHIFTREG[0..14]
SHIFTREG[0..15].CLKF = SCLOCK
SHIFTREG[0..15].RSTF = GND
SHIFTREG[0..15].SETF = GND

INV = CTR2[6] + (SHIFTREG[0] + SHIFTREG[1] + SHIFTREG[2]) * CTR1[2]

README = /SHIFTREG[0] * /SHIFTREG[1] * /SHIFTREG[2] * CTR1[2] * /CTR2[6]

DATAOUT[0..11] = SHIFTREG[4..15] * README

CTR1[0].T = /CTR1[2]
CTR1[1].T = CTR1[0] * /CTR1[2]
CTR1[2].T = CTR1[1] * CTR1[0] * /CTR1[2]
CTR1[0..2].CLKF = EXTCLOCK
CTR1[0..2].RSTF = SCLOCK
CTR1[0..2].SETF = GND
```

Table7.2 (continued)

68HC11 Program Listing

This section contains the code used in the HC11. This version averages sixteen data values.

```
;Program        Laser Interface Module
;Revision       1.0
;Programer      Brian Burgess
;Date           November 14, 1992
;
        org     $0000           ;This section contains RAM storage
        JMP     START           ;Jump to program in EPROM
SUM     DB      2               ;Storage for the sum
INA     DB      2               ;Storage for portA
TEMP    DB      1               ;Temporary Storage

NUMAVG  EQU     $10             ;Number of values to average
PORTA   EQU     $1000           ;Address of PORTA
PORTC   EQU     $1003           ;Address of PORTC
PORTB   EQU     $1004           ;Address of PORTB
PORTE   EQU     $100A           ;Address of PORTE
DDRC    EQU     $1007           ;Data Direction Register for PORTC
PACTL   EQU     $1026           ;PORTA Control Register

;       A data value is ready to be read, when bit 4 of Port C is low.
;       The data is 12 bits wide, with the most significant nibble in
;       bits 3-0 of Port C and the least significant byte in Port E.
;       This code polls Port C, when bit 4 is low, then the 4 least
;       significant bits of PortC already have the most significant
;       nibble of the data value to be read.  Next, the least
;       significant byte from PortE is read.  This value is now added to
;       a sum.  If the sum has the number of values to be averaged, then
;       the sum is shifted to the right the correct number of places
;       to produce the average of the values read.
        org     $E000           ;EPROM
START   LDAA    #$E0
        STAA    DDRC            ;Set up PortC for input
        LDAA    #0              ;Clear Accumulator A
        STAA    PORTC           ;Clear upper bits of PortC
        LDAA    #$3C
        STAA    DDRD            ;Set up PortD for output
;
AVG16   LDAA    PACTL           ; 4 cycles   set Port A bit 7 for output
        ORAA    #80             ;
        STAA    PACTL           ;
        LDD     #0              ; 3 cycles
        STD     SUM             ; 5 cycles
        LDX     #NUMAVG         ; 3 cycles
WTLOW   LDAA    PORTC           ; 4 cycles
        BITA    #$10            ; 2 cycles
        BEQ     WTLOW           ; 3 cycles
```

Table 7.2 (continued)

```
CTR2[0].T = /CTR2[6]
CTR2[1].T = CTR2[0] * /CTR2[6]
CTR2[2].T = CTR2[1] * CTR2[0] * /CTR2[6]
CTR2[3].T = CTR2[2] * CTR2[1] * CTR2[0] * /CTR2[6]
CTR2[4].T = CTR2[3] * CTR2[2] * CTR2[1] * CTR2[0] * /CTR2[6]
CTR2[5].T = CTR2[4] * CTR2[3] * CTR2[2] * CTR2[1] * CTR2[0] * /CTR2[6]
CTR2[6].T = CTR2[5] * CTR2[4] * CTR2[3] * CTR2[2] * CTR2[1] * CTR2[0] *
            /CTR2[6]
CTR2[0..6].CLKF = EXTCLOCK
CTR2[0..6].RSTF = SCLOCK
CTR2[0..6].SETF = GND

LED[1] = /INV
LED[2] = /INV * (LED[3] + DATAOUT[9] + DATAOUT[8] * DATAOUT[7] *
         (DATAOUT[6] + DATAOUT[5] + DATAOUT[4] * DATAOUT[3] *
         (DATAOUT[2] + DATAOUT[1])))
LED[3] = /INV * (DATAOUT[11] + DATAOUT[10] + DATAOUT[9] * DATAOUT[8] *
         (DATAOUT[7] + DATAOUT[6] + DATAOUT[5] * DATAOUT[4] *
         (DATAOUT[3] + DATAOUT[2])))
LED[4] = /INV * (DATAOUT[11] + DATAOUT[10] *(DATAOUT[9] + DATAOUT[8] *
         DATAOUT[7] * DATAOUT[6] * (DATAOUT[5] + DATAOUT[4] +
         DATAOUT[3] * DATAOUT[2] * DATAOUT[1])))
LED[5] = /INV * (DATAOUT[11] + DATAOUT[10] * DATAOUT[9] * (DATAOUT[8] +
         DATAOUT[7] + DATAOUT[6] * DATAOUT[5] * (DATAOUT[4] +
DATAOUT[3])))
LED[6] = /INV * DATAOUT[11]
LED[7] = /INV * (DATAOUT[11] * (DATAOUT[10] + DATAOUT[9] + DATAOUT[8] *
         DATAOUT[7] * (DATAOUT[6] + DATAOUT[5] + DATAOUT[4] *
         DATAOUT[3] * DATAOUT[2])))
LED[8] = /INV * (DATAOUT[11] * (DATAOUT[10] + DATAOUT[9] * DATAOUT[8] *
         (DATAOUT[7] + DATAOUT[6] + DATAOUT[5] * DATAOUT[4] *
         (DATAOUT[3] + DATAOUT[2] * DATAOUT[1]))))
LED[9] = /INV * (DATAOUT[11] * DATAOUT[10] * (DATAOUT[9] + DATAOUT[8] *
         DATAOUT[7] * DATAOUT[6] * (DATAOUT[5] + DATAOUT[4])))
LED[10] = /INV * (DATAOUT[11] * DATAOUT[10] * DATAOUT[9] * (DATAOUT[8] *
       DATAOUT[7] + DATAOUT[6] * DATAOUT[5] * (DATAOUT[4] + DATAOUT[3] *
         (DATAOUT[2] + DATAOUT[1]))))
```

Table 7.2 (continued)

```
;
POLL    LDAA    PORTC           ; 4 cycles   these 3 instruction wait
        BITA    #$10            ; 2 cycles   for README to be low.
        BNE     POLL            ; 3 cycles   port C already contains high
;                                            nibble, so don't re-read it.
        LDAB    PORTE           ; 4 cycles   now read low byte from portE.
        ADDD    SUM             ; 6 cycles   add it to the sum.
        STD     SUM             ; 5 cycles
        DEX                     ; 3 cycles   is it time to average?
        BNE     POLL            ; 3 cycles   if not, read next number.
        EORA    #$FF
        EORB    #$FF
        STAA    PORTB           ; 4 cycles
        STAB    PORTA           ; 4 cycles
        LDX     #NUMAVG         ; 3 cycles   output to port B, and now
        LDD     #0              ; 3 cycles   reset sum for next average.
        STD     SUM             ; 5 cycles
        BRA     POLL            ; 3 cycles   start reading numbers again.
        org $FFFE               ;Set up Reset Vector
        DW      START
        END
```

CHAPTER 8
SMART A to D MODULE

8.0 SA2D Overview

This section details the design of and serves as a reference for the Smart Analog to Digital (SA2D) converter board. The SA2D board allows multiple analog signals to be sampled at high frequencies by moving a significant portion of software control into hardware. The SA2D board replaces the normal software steps of sequencing a multiplexer through a set of analog signals and performing an A/D conversion on each. A write to the SA2D board automatically initiates the conversion of all analog inputs. The SA2D board returns a "complete" signal to the software to indicate when all conversions are complete. The software can then read the results of all A/D conversions.

The SA2D board is equipped with a bicolor Light Emitting Diode (LED) to indicate that the board is functioning. The LED alternates between the colors of red and green when the board is in operation.

8.1 Detailed Design\Design Components

This section provides the detailed design of the SA2D board. The following elements are implemented in the SA2 D board.

First In First Out (FIFO) Buffer - Provides software burst reads of previous A/D conversions while allowing the results of current A/D conversions to be stored. The result of each analog signal's A/D conversion is stored in the FIFO. The FIFO is eight (8) bits wide, so the results of each A/D conversion (16 bits) are stored by first writing the most significant byte then the least significant byte into the FIFO. The actual device used is an IDT7201. Reference the IDT Specialized Memories & Modules data book for details relating to this device.

A/D Converter – This device converts the magnitude of an analog signal to its equivalent digital form. The device used is a Burr Brown ADS7805P, providing sixteen (16) bits of resolution. Reference the Burr Brown IC Data Book – Data Conversion Products, for details relating to this device.

Analog Multiplexer – Routes one (1) of sixteen (16) different analog signals to the A/D converter. Two (2) Burr Brown MPC507 eight (8) channel, differential, multiplexers are cascaded to implement this function. Reference the Burr Brown IC Data Book – Data Conversion Products, for details relating to this device.

Instrumentation Amplifier – The differential outputs of the multiplexers are routed through an instrumentation amplifier to provide a single ended input to the A/D converter. The actual device used is a Burr Brown INA111BP. Reference the Burr Brown IC Data Book – Linear Products, for details relating to this device.

State Machine – The state machine is the heart of the SA2D board. It is responsible for the proper sequencing of the hardware. When the state machine recognizes a start indication, it sequences through the following steps for each analog signal: 1) select analog signal from the multiplexer, 2) perform A/D conversion, 3) Wait for conversion to complete, 4) store results of the conversion into the FIFO (MSB first then LSB). Once these steps are complete for all analog signals, the state machine returns a "complete" signal and waits for the next start indication. The state machine is implemented in an AMD PALCE610H-15 device. Reference the AMD PAL Devices data book for details relating to this device.

- Bicolor LED – The LED is provided to indicate that the SA2D board is operating. The LED alternates between the colors of red and green every time a sequence of conversions completes. One side of the LED is driven with the EOC signal while the other side is driven with the negated EOC signal.

8.2 State Machine Design

The SA2D state machine is responsible for the following steps:

1. Select Each Analog Channel of the Multiplexer
2. Wait for the multiplexer to settle
3. Start an A/D Conversion on the Selected Channel
4. Wait for A/D Conversion to Complete
5. Store the Conversion into the FIFO (MSB First, then LSB)
6. After the Last Channel is Converted then Return a "Complete" Indication.

The table lists the inputs and outputs of the SA2D state machine.

Table 8.1 State Machine I/O

Inputs	Outputs
RESET : SA2D Board Reset	ADDR : Selects Multiplexer Channel to Convert
START_LATCH : Latch Indicating the SA2D Board Should Start Converting All Channels	CONVERT : Initiates A/D Conversion
ADBUSY : Output of the A/D Converter Indicating That a Conversion is in Progress	FIFOW : Write Selected Byte of the A/D Conversion Into the FIFO

Inputs	Outputs
ADDR : Four Bit Counter Indicating Last Multiplexer Channel Converted	BYTE : Selects MSB or LSB of A/D Converted Data
MAXADDR : Hard Coded Number Indicating the Last Channel to Convert	EOC : Indicates the SA2D Board Has Completed the Conversion of All Channels

Table 8.1 (continued) State Machine I/O

The state diagram, Figure 8.1, defines the steps used in the design of the SA2D state machine. Since twelve (12) steps are used, a four (4) bit register must be used to store the current step. Steps 9, B, and D are omitted to minimize the number of bits that must be evaluated in the resulting equations.

For instance, by skipping these steps, evaluation of the least significant bit of the state register is not necessary when the most significant bit has a value of one (1) (true).

8.2.1 State Diagram Description

In the State Diagram of Figure 8.1 the text next to each state transition (arrow) provides an input expression that must be true for the transition to occur, followed by any synchronous output events. The required input expression is separated by the outputs by a colon (:). A forward slash (/) preceding a signal name indicates that the signal is negated (logic 0), while the absence of the slash indicates the signal is asserted. For instance, the transition from step 0 to step 1 occurs when the value of the START_LATCH is true. When this occurs, a logic one (1) is stored in a flip flop named CONVERT and the state machine transitions from state 0 to state 1. The transition from step 1 to step 2 occurs when inputs RESET and ADBUSY both have a value of logic zero (0). There are no corresponding outputs that are set during this transition.

Step 0: This is the beginning step of the state machine. The state machine waits in this step until a start signal is received. Once a start signal is received the first conversion is started on the A/D converter and control transitions to state 1. It should be noted here that a four (4) bit register named ADDR selects the current analog channel from the multiplexer. The ADDR register is cleared at reset and after converting the last channel, so it is safe to assume at this point that the correct (first) channel is already routed to the A/D converter.

Step 1: This step waits for the A/D converter to complete the conversion initiated during the transition from state 0. This is accomplished by monitoring the busy signal, named ADBUSY, from the A/D converter.

Step 2: When transitioning from this step to step 3, the address register is incremented to select the next signal on the multiplexer, and the MSB of the A/D converted data is written into the FIFO. The rational for incrementing the address register here rather than in the

transition from state 1 is to avoid the possibility of having the CONVERT signal momentarily asserted while selecting a new signal (although the probability is remote, the timing diagrams for the A/D converter do not guarantee it). Note that after incrementing the address register to select the next analog signal that the A/D should not be initiated for another 3.5 μSeconds to allow the signal to settle through the multiplexer and instrumentation amplifier. This time is computed for an accuracy of 0.01%. Since the frequency of the state machine's clock is 2.457 MHz this results in a delay of 9 states (must round up from 8.6). For this reason, the address is incremented as early as possible. States 3 through E provide the 9 state settling time delay. The remaining non-time critical functionality is spread across these remaining steps for clarity.

Step 3: During the transition from this step to step 4, the address register is checked to determine if all channels have already been converted. If so, the address register is reset such that the first analog channel is already selected upon entering state 0. Note that the diagram does not transition back to state 0 immediately after recognizing that all channels have been converted. This ensures that the first channel has settled prior to entering state 0, just in case another sequence has already been initiated.

Step 4: When transitioning from this step to step 5, the LSB is selected from the A/D converter and routed to the FIFO. Asserting the BYTE signal to the A/D converter selects the LSB.

Step 5: When transitioning from this step to step 6, the LSB of the converted data is written into the FIFO (follows the MSB). The BYTE signal is maintained in the asserted state during this write.

Step 6 and 7: During the transitions from step 6 through step 7 and into step 8, the BYTE signal is maintained in the asserted state. The BYTE signal must be asserted as long as the FIFOW signal is asserted and is maintained asserted for an additional state to avoid a race condition (do not want to negate BYTE and FIFOW concurrently). The BYTE signal remains asserted into step 8 simply to minimize the logic required in the generation of the signal.

Step 8: During the transition from step 8 to step 9, the address register is evaluated to determine if all channels have been converted. If so, the EOC signal is toggled to provide an indication to the host processor that the entire conversion process is complete. Since it is possible that the host processor could recognize the completion flag (EOC) and initiate another sequence of conversions before transitioning back to state 0, the START_LATCH signal is latched.

Steps A and C: These steps only exist to provide the required settling time (see the description in step 2).

Step E: Transition from this step back to step 0 occurs if all channels have been converted (ADDR==0). If all channels have not been converted, then a transition back to step 1 occurs and an A/D conversion is immediately initiated on the selected analog channel.

Smart A to D State Diagram

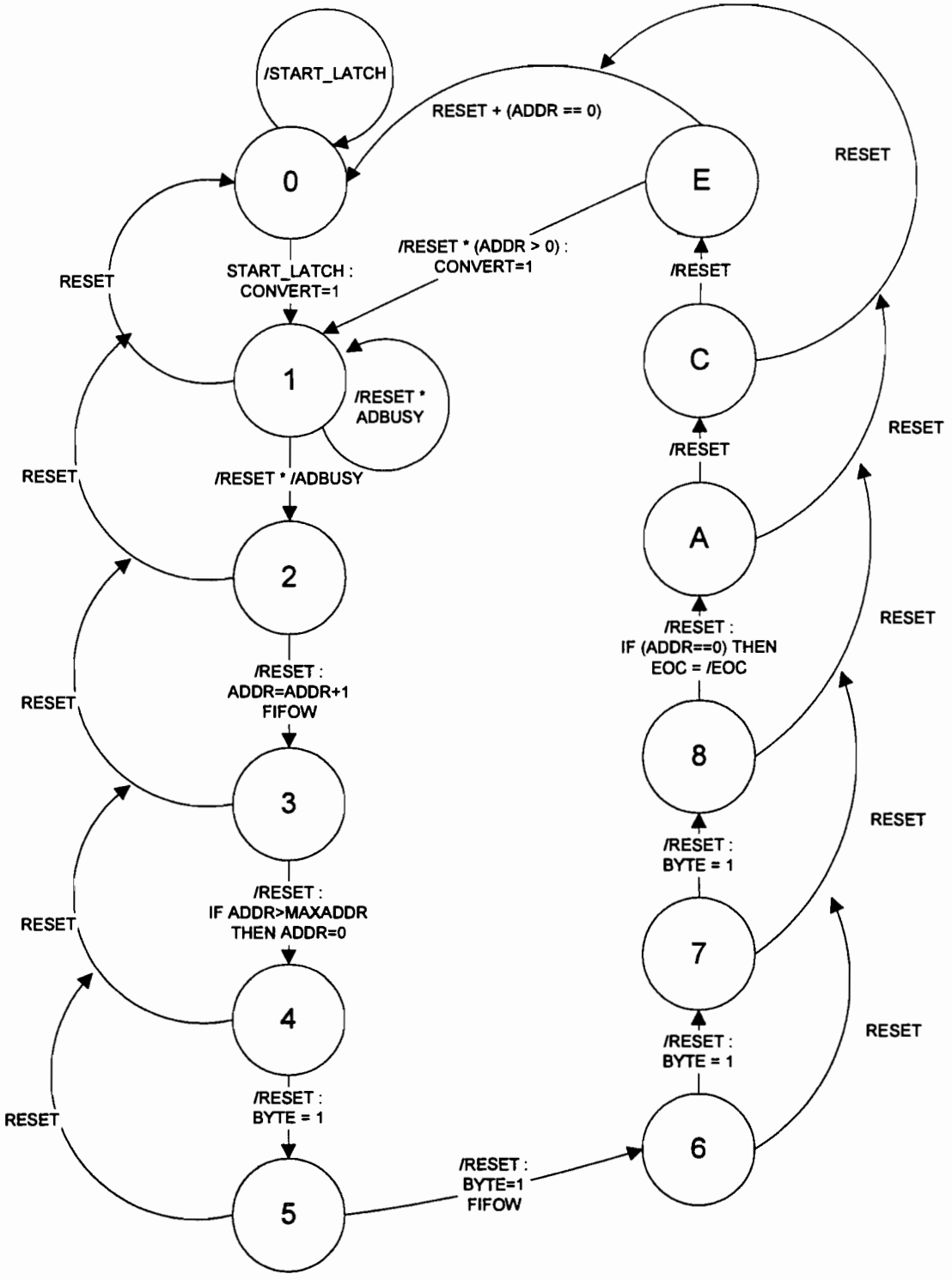


Figure 8.1 Smart A to D State Diagram

8.3 Hardware Installation

Power Connections

Power to the SA2D board is applied at terminal block J3. The board requires 5V DC and +/- 15V DC supplies. The ground of the 5V supply is connected to DGND (digital ground). The ground for the +/- 15V supply is connected to AGND (analog ground). These two grounds must be connected externally at a single point. If this point does not already exist in the system then adding a wire between AGND and DGND can connect the two grounds.

Analog Signals

Analog signals are connected to the SA2D board at terminal block J1. The connection for each analog signal is differential. The high side of the signal should be connected to the terminal labeled + while the low side is connected to the – terminal. The connection for channel 1 is labeled Ch1, channel 2 as Ch2, etc. When reading the converted values from the FIFO, channel 1 is read first while the last channel is read last. The number of channels that are converted depends on the hard coded value in the state machine (PAL).

Processor Interface

The host processor is connected to the SA2D board with a 26-pin DIN connector at J2. The schematic in the attached appendix defines the pin-out of this connector.

Following this page Table 8.2 gives the Pal equations, and then Figures 8.2 through Figures 8.5 show the board schematics.

Table 8.2 Pal Equations

```
;PALASM Design Description

;---------------------------------- Declaration Segment ------------
TITLE    Smart Analog to Digital Board Controller
PATTERN

CHIP  _SMARTA2D  PALCE610

;---------------------------------- PIN Declarations ---------------
PIN  1          CLK1                                    COMBINATORIAL ;
INPUT
PIN  13         CLK2                                    COMBINATORIAL ;
INPUT
PIN  2          /RESET                                  COMBINATORIAL ;
INPUT
PIN  3..5       ST[0..2]                                REGISTERED ; OUTPUT
PIN  6          START_LATCH                             COMBINATORIAL ;
OUTPUT
PIN  7          EOC_INV                                 COMBINATORIAL ;
OUTPUT
PIN  8          EOC                                     REGISTERED ; OUTPUT
PIN 9           ST[3]                                   REGISTERED
PIN  10         BYTE                                    REGISTERED ; OUTPUT
PIN  11         /A2DCS                                  COMBINATORIAL ;
INPUT
; PIN  12       GROUND
PIN  14         READ                                    COMBINATORIAL ;
INPUT
PIN  15         A3_INV                                  COMBINATORIAL ;
OUTPUT
PIN  16..19     ADDR[3..0]                              REGISTERED ; OUTPUT
PIN  20         /CONVERT                                REGISTERED ; OUTPUT
PIN  21         /FIFOW                                  REGISTERED ; OUTPUT
PIN  22         /FIFOR                                  COMBINATORIAL ;
OUTPUT
PIN  23         /ADBUSY                                 COMBINATORIAL ;
INPUT

STRING RESTART '(/ADDR[3]*/ADDR[2]*/ADDR[1]*/ADDR[0])'
STRING MAXCNT '(ADDR[3])'  ; After 0-7 conversions count=8, so reset
counter
;---------------------------------- Boolean Equation Segment ------
EQUATIONS

CASE ST[3..0]
BEGIN
0:
  BEGIN
    IF /START_LATCH THEN
      BEGIN
        ST[3..0]=0
      END
    ELSE
      BEGIN
```

```
        ST[3..0]=1
      END
  END

1:
  BEGIN
    IF RESET THEN
      BEGIN
        ST[3..0]=0
      END
    IF (ADBUSY * /RESET) THEN
      BEGIN
        ST[3..0]=1
      END
    IF (/ADBUSY * /RESET) THEN
      BEGIN
        ST[3..0]=2
      END
  END

2:
  BEGIN
    IF /RESET THEN
      BEGIN
        ST[3..0]=3
      END
    ELSE
      BEGIN
        ST[3..0]=0
      END
  END

3:
  BEGIN
    IF /RESET THEN
      BEGIN
        ST[3..0]=4
      END
    ELSE
      BEGIN
        ST[3..0]=0
      END
  END

4:
  BEGIN
    IF /RESET THEN
      BEGIN
        ST[3..0]=5
      END
    ELSE
      BEGIN
        ST[3..0]=0
      END
  END

5:
  BEGIN
    IF /RESET THEN
```

```
      BEGIN
        ST[3..0]=6
      END
    ELSE
      BEGIN
        ST[3..0]=0
      END
  END

6:
  BEGIN
    IF /RESET THEN
      BEGIN
        ST[3..0]=7
      END
    ELSE
      BEGIN
        ST[3..0]=0
      END
  END


7:
  BEGIN
    IF /RESET THEN
      BEGIN
        ST[3..0]=8
      END
    ELSE
      BEGIN
        ST[3..0]=0
      END
  END


8:
  BEGIN
    IF /RESET THEN
      BEGIN
        ST[3..0]=10
      END
    ELSE
      BEGIN
        ST[3..0]=0
      END
  END

10:
  BEGIN
    IF /RESET THEN
      BEGIN
        ST[3..0]=12
      END
    ELSE
      BEGIN
        ST[3..0]=0
      END
  END
```

```
12:

  BEGIN
    IF /RESET THEN
      BEGIN
        ST[3..0]=14
      END
    ELSE
      BEGIN
        ST[3..0]=0
      END
  END


14:
  BEGIN
    IF RESTART + RESET THEN
      BEGIN
        ST[3..0]=0
      END
    IF /RESET * /RESTART THEN
      BEGIN
        ST[3..0]=1
      END
  END


OTHERWISE:
  BEGIN
    ST[3..0]=0
  END

END
ST[3..0].CLKF=CLK1
ST[3..0].RSTF=GND


START_LATCH=/RESET*(A2DCS*/READ+START_LATCH*/ST[0])

FIFOR = A2DCS * READ


CONVERT=/RESET*(/ST[3]*/ST[2]*/ST[1]*/ST[0]*START_LATCH+
                ST[3]*ST[2]*ST[1]*/ST[0]*/(RESTART) )
CONVERT.CLKF=CLK2
CONVERT.RSTF=GND

EOC.T=/RESET*ST[3]*/ST[2]*/ST[1]*/ST[0]*(RESTART)
EOC.CLKF=CLK1
EOC.RSTF=GND

BYTE=/RESET*/ST[3]*ST[2]
BYTE.CLKF=CLK1
BYTE.RSTF=GND

FIFOW=/RESET*(/ST[3]*/ST[2]*ST[1]*/ST[0] + /ST[3]*ST[2]*/ST[1]*ST[0])
```

```
FIFOW.CLKF=CLK2
FIFOW.RSTF=GND

; The following equations had to be derived manually, due to a bug in
; PALASM.  The equations for the address signals remain in the case
statement
; of the state machine, but are commented out.  The following equations
; are derived from those comments.
; Each address signal is derived as follows:
;   The first line of the equation basically resets the address signal to
;   0 if reset is asserted or an invalid state is detected.
;   The second term resets the address to 0 if the maximum count is
exceeded
;   (see the commented equation in state 4).
;   The final term increments the address after state 3.

ADDR[3].T=RESET*ADDR[3]
          + /ST[3]*/ST[2]*ST[1]*ST[0]*(MAXCNT)*ADDR[3]
          + /ST[3]*/ST[2]*ST[1]*/ST[0]*ADDR[2]*ADDR[1]*ADDR[0]
ADDR[2].T=RESET*ADDR[2]
          + /ST[3]*/ST[2]*ST[1]*ST[0]*(MAXCNT)*ADDR[2]
          + /ST[3]*/ST[2]*ST[1]*/ST[0]*ADDR[1]*ADDR[0]
ADDR[1].T=RESET*ADDR[1]
          + /ST[3]*/ST[2]*ST[1]*ST[0]*(MAXCNT)*ADDR[1]
          + /ST[3]*/ST[2]*ST[1]*/ST[0]*ADDR[0]
ADDR[0].T=RESET*ADDR[0]
          + /ST[3]*/ST[2]*ST[1]*ST[0]*(MAXCNT)*ADDR[0]
          + /ST[3]*/ST[2]*ST[1]*/ST[0]

A3_INV = /ADDR[3]

ADDR[3..0].CLKF=CLK2
ADDR[3..0].RSTF=GND

EOC_INV=/EOC

;---------------------------------- Simulation Segment ------------
SIMULATION
TRACE_ON /EOC /RESET CLK1 CLK2 /ADBUSY ST[3..0] ADDR[3..0] START_LATCH
/A2DCS READ BYTE /FIFOW /CONVERT
SETF A2DCS /READ
SETF /A2DCS
SETF /ADBUSY RESET
CLOCKF CLK1 CLK2
SETF /RESET
CLOCKF CLK1 CLK2
SETF A2DCS /READ
CLOCKF CLK1 CLK2
SETF /A2DCS
CLOCKF CLK1 CLK2
CLOCKF CLK1 CLK2
CLOCKF CLK1 CLK2
CLOCKF CLK1 CLK2
CLOCKF CLK1 CLK2
CLOCKF CLK1 CLK2
CLOCKF CLK1 CLK2
CLOCKF CLK1 CLK2
```

```
CLOCKF CLK1 CLK2
CLOCKF CLK1 CLK2
CLOCKF CLK1 CLK2
CLOCKF CLK1 CLK2
CLOCKF CLK1 CLK2
CLOCKF CLK1 CLK2
CLOCKF CLK1 CLK2
CLOCKF CLK1 CLK2
CLOCKF CLK1 CLK2
CLOCKF CLK1 CLK2
CLOCKF CLK1 CLK2
CLOCKF CLK1 CLK2
CLOCKF CLK1 CLK2
CLOCKF CLK1 CLK2
CLOCKF CLK1 CLK2
CLOCKF CLK1 CLK2
CLOCKF CLK1 CLK2
CLOCKF CLK1 CLK2
CLOCKF CLK1 CLK2
CLOCKF CLK1 CLK2
CLOCKF CLK1 CLK2
CLOCKF CLK1 CLK2
CLOCKF CLK1 CLK2
CLOCKF CLK1 CLK2
CLOCKF CLK1 CLK2
CLOCKF CLK1 CLK2
CLOCKF CLK1 CLK2
CLOCKF CLK1 CLK2
CLOCKF CLK1 CLK2
CLOCKF CLK1 CLK2
CLOCKF CLK1 CLK2
CLOCKF CLK1 CLK2
CLOCKF CLK1 CLK2
CLOCKF CLK1 CLK2
CLOCKF CLK1 CLK2
CLOCKF CLK1 CLK2
CLOCKF CLK1 CLK2
CLOCKF CLK1 CLK2
CLOCKF CLK1 CLK2
CLOCKF CLK1 CLK2
CLOCKF CLK1 CLK2
CLOCKF CLK1 CLK2
CLOCKF CLK1 CLK2
CLOCKF CLK1 CLK2
CLOCKF CLK1 CLK2
CLOCKF CLK1 CLK2
CLOCKF CLK1 CLK2
CLOCKF CLK1 CLK2
CLOCKF CLK1 CLK2
```

A/D BOARD

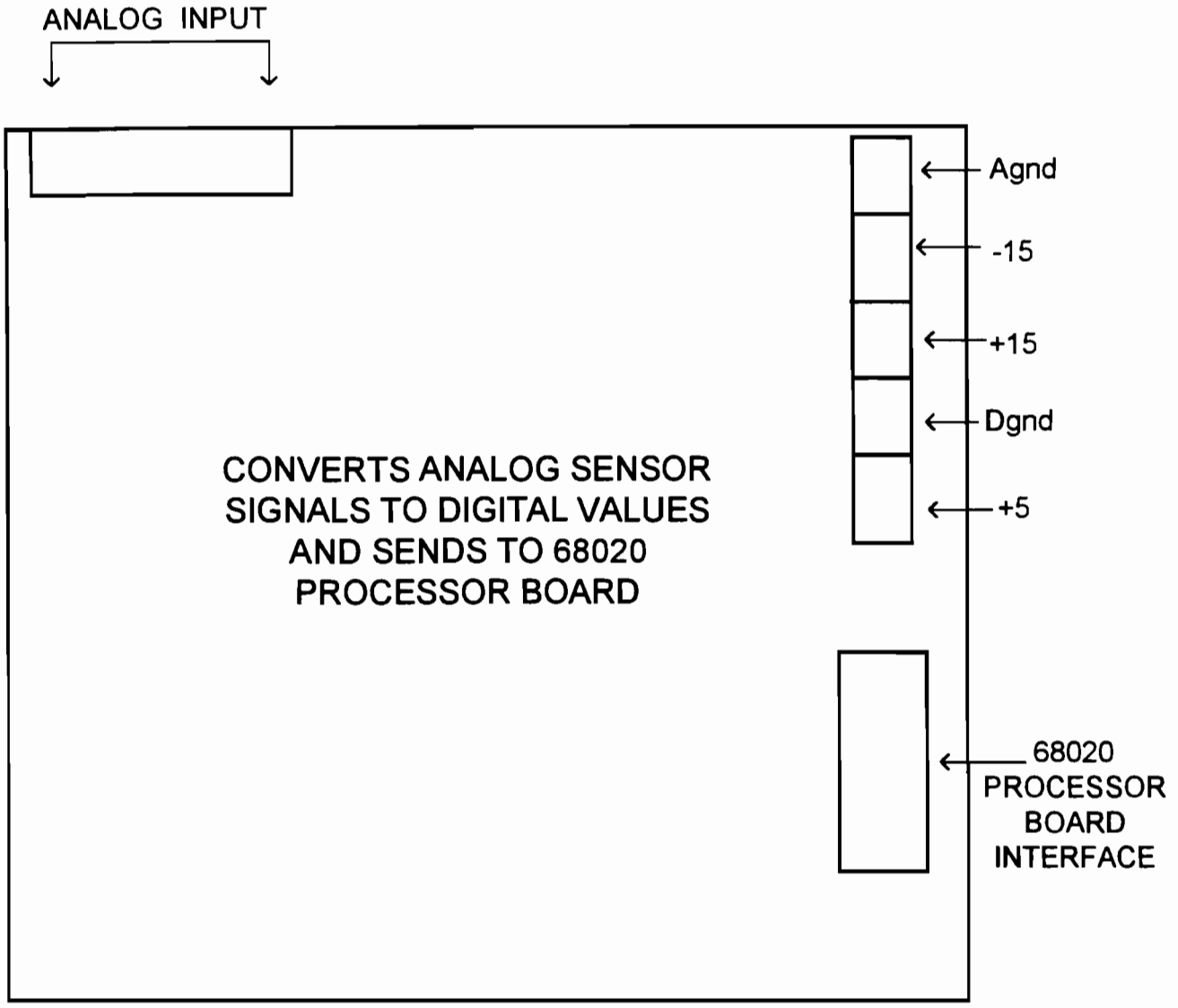


Figure 8.2 A/D Board Layout

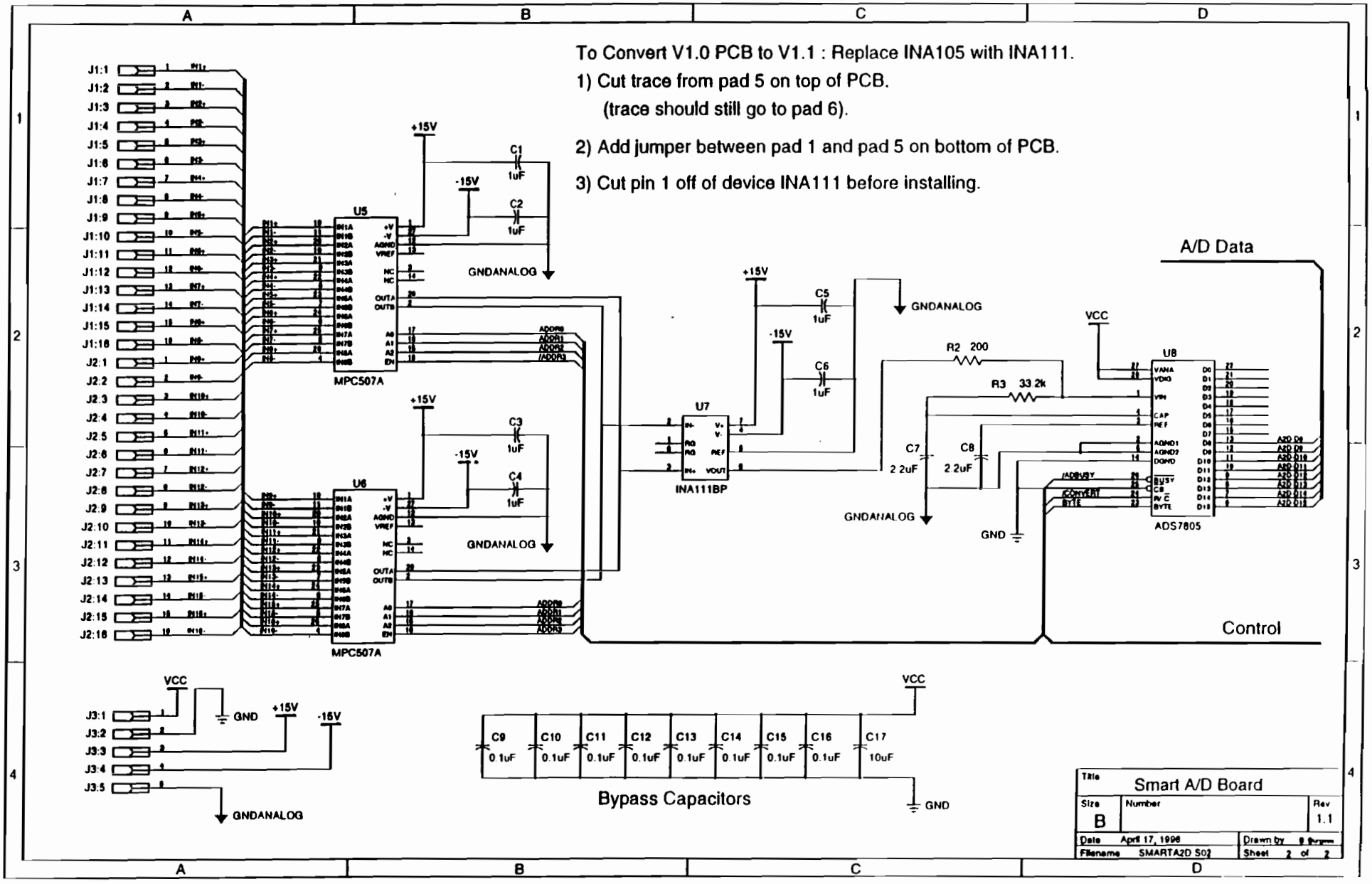


Figure 8.3 Schematic Smart A/D Board 1

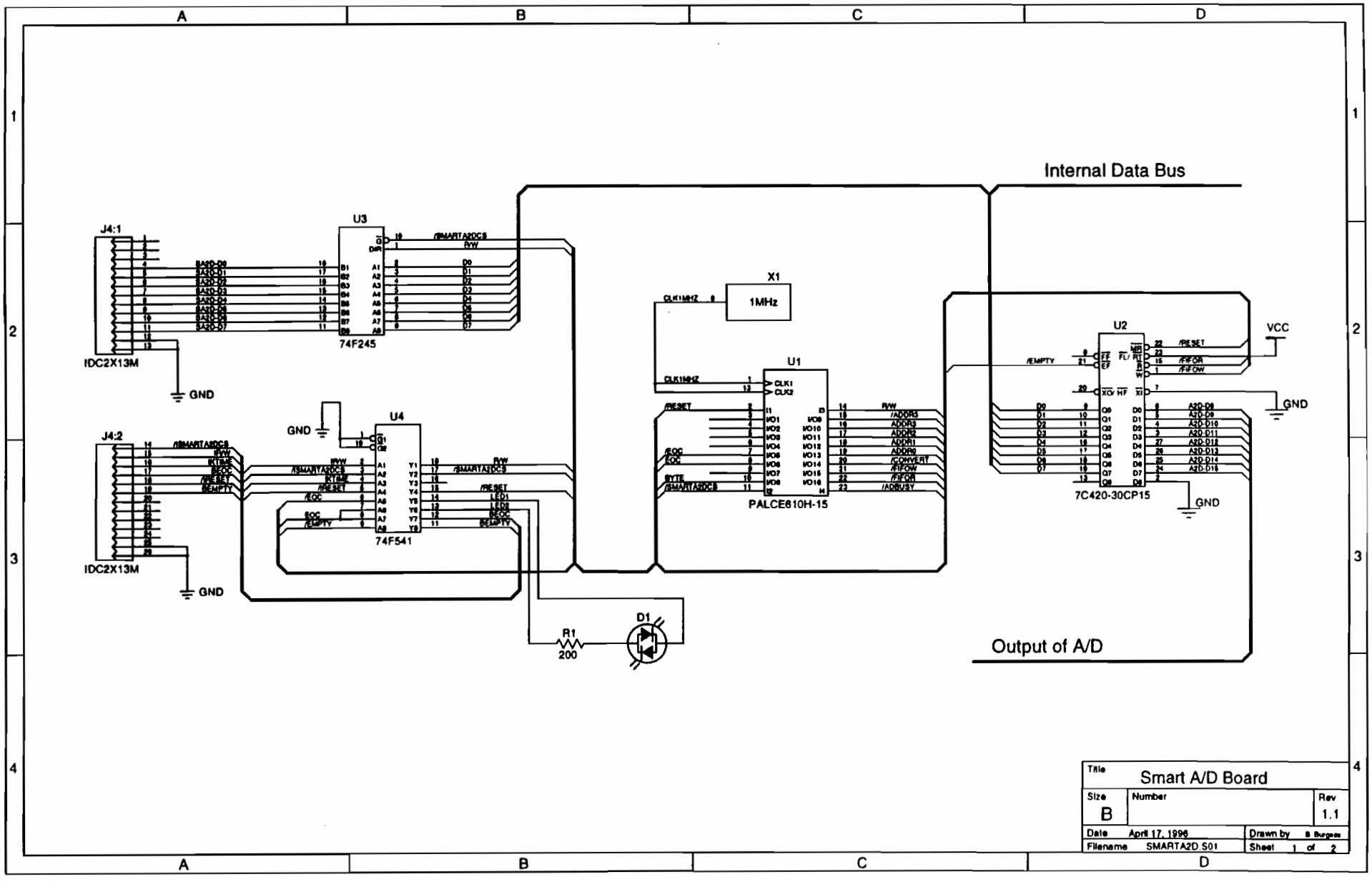


Figure 8.4 Schematic Smart A/D Board 2

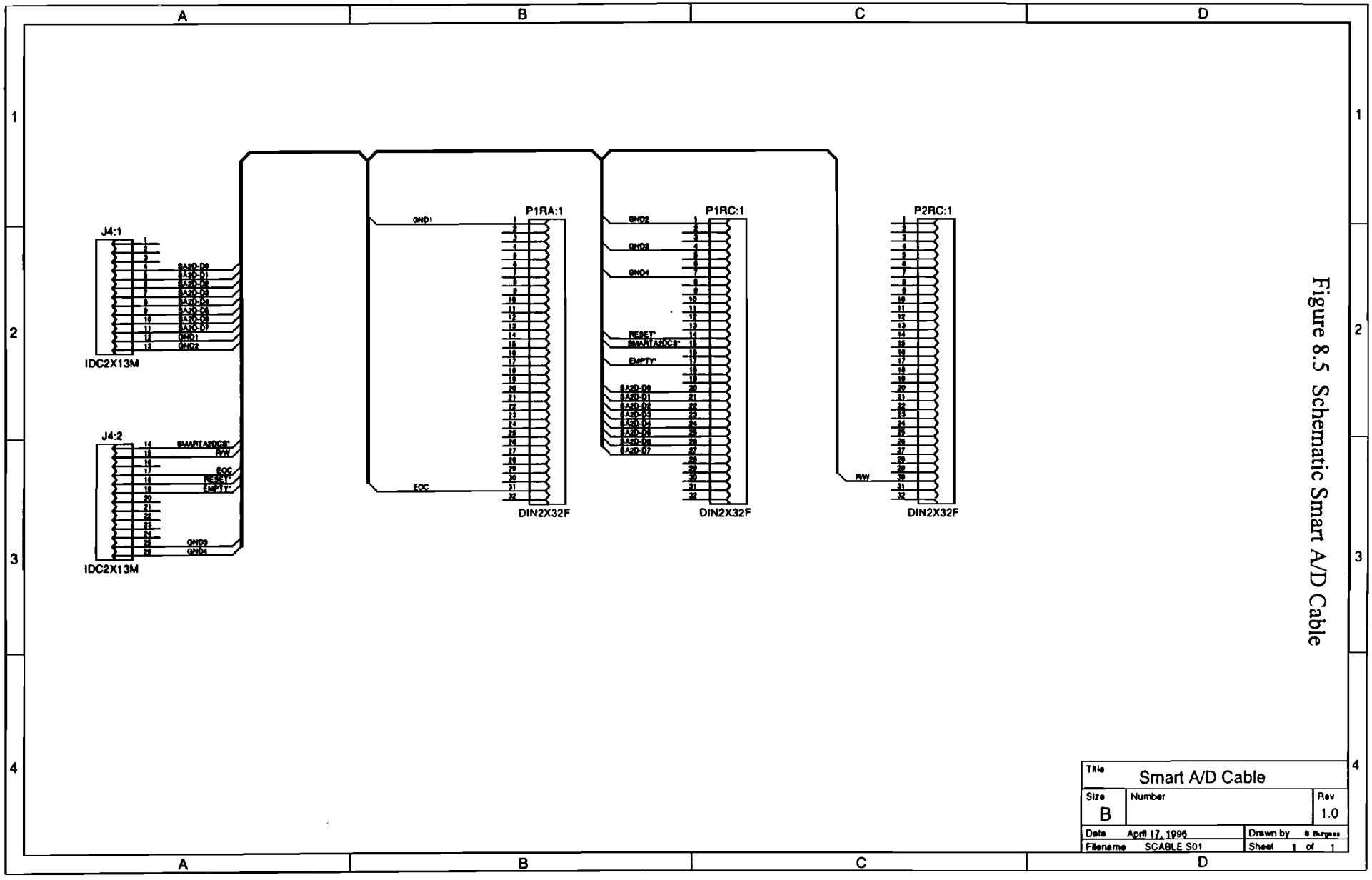


Figure 8.5 Schematic Smart A/D Cable

CHAPTER 9
The 68020 MODULE (K Processor)

9.0 Overview

This section provides a description of the 68020 Processor module or K Processor board. This board is used to obtain the data from the various sensors and either process this data in real-time or sends it to the PC. The communication protocol between this board and the PC is provided in an earlier chapter. The processor will permit the interface of up to 16 analog devices, or nine acoustic sensors which are designed to interface with the on board Intel 83C54 timers. The processor, when in the real-time computation mode, will compute profile and/or rut from either laser using the 'string line method', raw readings from acoustic sensors. An overall block diagram of the basic board is illustrated in Figure 9.1, 9.2, and 9.3. The board provides 64K bytes of EPROM and 32 K bytes of static RAM. Provisions are included for 2 12 bit A/D converters, although they are no longer used with the smart A/D module. The smart A/D and other smart boards designed by project personnel can access the board via a direct linkage to the upper lines of the 32 bit data bus. The board contains four 82C54 timers which provide direct interface for up to ten acoustic sensors. Two counters are used to control the A/D sampling, for implementing the South Dakota profiling procedure, and one for a simulated distance signal (see time/distance simulation switch in chapter 4). Two MC68230 parallel ports are included for general parallel and interrupt interface.

The memory map for the various modules is provided in Table 9.1. Table 9.2 provides the interrupt assignments. Tables 9.3 to 9.8 provide the basic pal equations for the 6 Pals used for address decoding and other control circuitry. Table 9.10 and 9.11 provide a pin out of the back plane connections to the 68020 module. Figure 9.4 to 9.11 show the schematics of the board.

One additional comment should be noted regarding the connections of the acoustic to the Amphenol connections on the main chassis module in Chapter 4. The two 82C54 IC timers at U36 and U37 (Figure 9.8) are used for the acoustic timers. Each IC has 3 timers, IC timer 0 thru 2. The signals on each timer are sent to the P2RA (P2 connector, A row, see Figure 9.11) on the K board. Each P1 and P2 connector on the board plugs into the J1 and J2 female connectors.

Pin 11, 14 and 16 on U36 (top timer) and U37, second timer.

IC Timer 0 is used for the distance simulation. IC Timer 1 and 2 on U36 are used for acoustics 2 and 3. IC Timer 0 and 1 on U37 are used for acoustic timer 4 and 5. IC timer 2 on U37. is used for acoustic timer 1

The appropriate pins on the P2RA connector for each of the acoustic timers are connected to the connector on the back panel (see Tables 4.7 and 4.8 in Chapter 4).

The IC to P2RA pin outs are given as follows:

acoustic 1 - U37 pin 16 P2RA - 24
acoustic 2 - U36 pin 14 P2RA - 20
acoustic 3 - U36 pin 16 P2RA - 21
acoustic 4 - U37 pin 11 P2RA - 22
acoustic 5 - U37 pin 14 P2RA - 23

Table 9.1 Memory Map

DEVICE TYPE	DESCRIPTION	ADDRESS	RANGE
		From	To
ROM 0	ROM AREA	$ 0000 : 0000	$ 0000 : FFFF
ROM 1	ROM AREA	$ 0001 : 0000	$ 0001 : FFFF
ROM 2	ROM AREA	$ 0002 : 0000	$ 0002 : FFFF
ROM 3	ROM AREA	$ 0003 : 0000	$ 0003 : FFFF
NOT USED	EXPANSION AREA	$ 0004 : 0000	3 FFF : FFFF
68681 DUART	68681 DUART (16 Bytes)	$ 4000 : 0000	$ 4000 : 000F
		$ 0004 : 0010	$ 0004 : 7FFF
NOT USED	EXPANSION AREA		
68230_1 PI/T	68230 PI/T (32 BYTES)	$ 4000 : 8000	$ 4000 : 801F
68230_2 PI/T	68230 PI/T (32 BYTES)	$ 4001 : 0000	$ 4001 : 001F
HS 9412_1		$ 4001 : 8000	
HS 9412_2		$ 4002 : 0000	
8254 TIMER_1	8254 TIMER_1	$ 4002 : 8000	$ 4002 : 8003
8254 TIMER_2	8254 TIMER_2	$ 4003 : 0000	$ 4003 : 0003
8254 TIMER_3	8254 TIMER_3	$ 4003 : D000	$ 4003 : D3FF
8254 TIMER_4	8254 TIMER_4	$ 4003 : D400	$ 4003 : D7FF
SMART A/D	SMART A/D	$ 4003 : C000	$ 4003 : C01E
NOT USED	EXPANSION AREA	$ 4003 : 8004	7FFF : FFFF
RAM 0	RAM AREA	$ 8000 : 0000	$ 8000 : 7FFF
RAM 1	RAM AREA	$ 8000 : 8000	$ 8000 : FFFF
RAM 2	RAM AREA	$ 8001 : 0000	$ 8001 : 7FFF
RAM 3	RAM AREA	$ 8001 : 8000	$ 8001 : FFFF

Table 9.2 Interrupt Assignment

Interrupt Level	Source Devices	IRQ Type
1		
2		
3	PI/T 68230_2 Timer (TIMOUT signal)	Autovector Interrupt
4	DUART 68681	IRQ 4
5	PI/T 68230_1	IRQ 5
6	PI/T 68230_2	IRQ 6

Table 9.3 PAL U13 Equations

```
□
;PALASM Design Description

;---------------------------------- Declaration Segment ------------
TITLE    EMCPAL1.PDS
PATTERN  A
REVISION 1.0
CHIP   EMCPAL1   PAL22V10

;---------------------------------- PIN Declarations ---------------
PIN  1          A00                  COMBINATORIAL              ; INPUT
PIN  2          A01                  COMBINATORIAL              ; INPUT
PIN  3          A17                  COMBINATORIAL              ; INPUT
PIN  4          A30                  COMBINATORIAL              ; INPUT
PIN  5          A31                  COMBINATORIAL              ; INPUT
PIN  6          SEG1                 COMBINATORIAL              ; INPUT
PIN  7          CPUSP                COMBINATORIAL              ; INPUT
PIN  8          /AS                  COMBINATORIAL              ; INPUT
PIN  9          WAIT2                COMBINATORIAL              ; INPUT
PIN  10         R                    COMBINATORIAL              ; INPUT
PIN  11         SIZ0                 COMBINATORIAL              ; INPUT
PIN  12         GND                                             ; INPUT
PIN  13         SIZ1                 COMBINATORIAL              ; INPUT
PIN  14         MDSACK1              COMBINATORIAL              ; OUTPUT
PIN  15         MDSACK0              COMBINATORIAL              ; OUTPUT
PIN  16         /ROM3                COMBINATORIAL              ; OUTPUT
PIN  17         /ROM2                COMBINATORIAL              ; OUTPUT
PIN  18         /ROM1                COMBINATORIAL              ; OUTPUT
PIN  19         /ROM0                COMBINATORIAL              ; OUTPUT
PIN  20         /RAM3                COMBINATORIAL              ; OUTPUT
PIN  21         /RAM2                COMBINATORIAL              ; OUTPUT
PIN  22         /RAM1                COMBINATORIAL              ; OUTPUT
PIN  23         /RAM0                COMBINATORIAL              ; OUTPUT
PIN  24         VCC                                             ; INPUT
```

```
;OUTPUT EQUATIONS----------------------
EQUATIONS

RAM0 = AS * /CPUSP * SEG1 * A31 * /A30 * /A17 * /A01 * /A00

RAM1 = AS * /CPUSP * SEG1 * A31 * /A30 * /A17 *
       ( /A01 * A00 + /A01 * SIZ1 + /A01 * /SIZ0 )

RAM2 = AS * /CPUSP * SEG1 * A31 * /A30 * /A17 *
       ( A01 * /A00 + /A01 * /SIZ1 * /SIZ0 + /A01 * SIZ1 * SIZ0 +
         /A01 * A00 * SIZ1 )

RAM3 = AS * /CPUSP * SEG1 * A31 * /A30 * /A17 *
       ( A01 * A00 + /SIZ1 * /SIZ0 + A01 * SIZ1 + A00 * SIZ1 * SIZ0 )

ROM0 = R * AS * /CPUSP * SEG1 * /A31 * /A30 * /A17 * /A01 * /A00

ROM1 = R * AS * /CPUSP * SEG1 * /A31 * /A30 * /A17 *
       ( /A01 * A00 + /A01 * SIZ1 + /A01 * /SIZ0 )

ROM2 = R * AS * /CPUSP * SEG1 * /A31 * /A30 * /A17 *
       ( A01 * /A00 + /A01 * /SIZ1 * /SIZ0 + /A01 * SIZ1 * SIZ0 +
         /A01 * A00 * SIZ1 )

ROM3 = R * AS * /CPUSP * SEG1 * /A31 * /A30 * /A17 *
       ( A01 * A00 + /SIZ1 * /SIZ0 + A01 * SIZ1 + A00 * SIZ1 * SIZ0 )

MDSACK0 = AS * /CPUSP * SEG1 * (/A17 * A31 * /A30 + R * WAIT2 * /A31 *
/A30)

MDSACK1 = AS * /CPUSP * SEG1 * (/A17 * A31 * /A30 + R * WAIT2 * /A31 *
/A30 )
```

Table 9.4 PAL U14 Equations

```
;PALASM Design Description
□

□
;---------------------------------- Declaration Segment ------------
□
TITLE    K processor peripheral decoder
□
PATTERN  A
□
REVISION 1.0
□
CHIP   KPALU14   PAL22V10
□
```

```
□
;---------------------------------- PIN Declarations ---------------
PIN  1          A14                  COMBINATORIAL              ; INPUT
;PIN  2          /SMADCS              COMBINATORIAL              ; OUTPUT
PIN  3          A17                  COMBINATORIAL              ; INPUT
PIN  4          A30                  COMBINATORIAL              ; INPUT
PIN  5          A31                  COMBINATORIAL              ; INPUT
PIN  6          SEG1                 COMBINATORIAL              ; INPUT
PIN  7          CPUSP                COMBINATORIAL              ; INPUT
PIN  8          /AS                  COMBINATORIAL              ; INPUT
PIN  9          /DS                  COMBINATORIAL              ; INPUT
PIN  10         R                    COMBINATORIAL              ; INPUT
PIN  11         SIZ0                 COMBINATORIAL              ; INPUT
PIN  12         GND                                             ; INPUT
PIN  13         SIZ1                 COMBINATORIAL              ; INPUT
PIN  14         A16                  COMBINATORIAL              ; INPUT
PIN  15         A15                  COMBINATORIAL              ; INPUT
PIN  16         AD_CS                COMBINATORIAL              ; OUTPUT
PIN  17         AD1_NAD2             COMBINATORIAL              ; OUTPUT
PIN  18         /8254_1CS            COMBINATORIAL              ; OUTPUT
PIN  19         /8254_2CS            COMBINATORIAL              ; OUTPUT
PIN  20         /SMADCS              COMBINATORIAL              ; OUTPUT
PIN  21         /230_2CS             COMBINATORIAL              ; OUTPUT
PIN  22         /230_1CS             COMBINATORIAL              ; OUTPUT
PIN  23         /681CS               COMBINATORIAL              ; OUTPUT
PIN  24         VCC                                             ; INPUT

;OUTPUT EQUATIONS----------------------
EQUATIONS

681CS = AS * /CPUSP * SEG1 * /A31 * A30 * /A17 * /A16 * /A15

230_1CS = DS * AS * /CPUSP * SEG1 * /A31 * A30 * /A17 * /A16 * A15

230_2CS = DS * AS * /CPUSP * SEG1 * /A31 * A30 * /A17 * A16 * /A15

AD_CS = R * DS * AS * /CPUSP * SEG1 * /A31 * A30 *
       (/A17 * A16 * A15 + A17 * /A16 * /A15)

AD1_NAD2 = R * DS * AS * /CPUSP * SEG1 * /A31 * A30 * /A17 * A16 * A15

8254_1CS = DS * AS * /CPUSP * SEG1 * /A31 * A30 * A17 * /A16 * A15

8254_2CS = DS * AS * /CPUSP * SEG1 * /A31 * A30 * A17 * A16 * /A15

SMADCS = DS * AS * /CPUSP * SEG1 * /A31 * A30 * A17 * A16 * A15 * A14
```

Table 9.5 PAL U15 Equations

```
□
;PALASM Design Description
□

□
;---------------------------------- Declaration Segment ------------
TITLE    SEGMENT DECODER PAL
```

```
PATTERN  A
REVISION 1.0

CHIP   KPALU15   PAL22V10

;--------------------------------- PIN Declarations ---------------
PIN  1          A13                COMBINATORIAL            ; INPUT
PIN  2          A14                COMBINATORIAL            ; INPUT
PIN  3          A15                COMBINATORIAL            ; INPUT
PIN  4          A16                COMBINATORIAL            ; INPUT
PIN  5          A17                COMBINATORIAL            ; INPUT
PIN  6          A18                COMBINATORIAL            ; INPUT
PIN  7          A19                COMBINATORIAL            ; INPUT
PIN  8          X                  COMBINATORIAL            ; INPUT
PIN  9          Y                  COMBINATORIAL            ; INPUT
PIN  10         FC0                COMBINATORIAL            ; INPUT
PIN  11         FC1                COMBINATORIAL            ; INPUT
PIN  12         GND                                         ; INPUT
PIN  13         FC2                COMBINATORIAL            ; INPUT
PIN  14         /AS                COMBINATORIAL            ; INPUT
PIN  15         WAIT2              COMBINATORIAL            ; INPUT
PIN  16         /8254_1CS          COMBINATORIAL            ; INPUT
PIN  17         /8254_2CS          COMBINATORIAL            ; INPUT
;PIN  18         /8254_3CS          COMBINATORIAL            ; INPUT
PIN  19         8254_DSACK         COMBINATORIAL            ; OUTPUT
PIN  20         IACK               COMBINATORIAL            ; OUTPUT
PIN  21         /COP1CS            COMBINATORIAL            ; OUTPUT
PIN  22         CPUSP              COMBINATORIAL            ; OUTPUT
PIN  23         SEG1               COMBINATORIAL            ; OUTPUT
PIN  24         VCC                                         ; INPUT

;OUTPUT EQUATIONS----------------------
EQUATIONS

IACK = FC2 * FC1 * FC0 * A19 * A18 * A17 * A16 * AS

COP1CS = FC2 * FC1 * FC0 * /A19 * /A18 * A17 * /A16 *
         /A15 * /A14 * A13 * AS

CPUSP = FC2 * FC1 * FC0

SEG1 = X * Y * /A19 * /A18

8254_DSACK = WAIT2 * (8254_1CS + 8254_2CS)
□
```

Table 9.6 PAL U33 Equations

```
;PALASM Design Description
□

□
;--------------------------------- Declaration Segment ------------
```

```
TITLE    K processor U33
PATTERN  1-a
REVISION a

CHIP  kpal_u33  PALCE26V12
;---------------------------------- PIN Declarations ---------------
PIN  1          CLK16                                   COMBINATORIAL ;
INPUT
PIN  4          SEG1                                    COMBINATORIAL ;
INPUT
PIN  5          CPUSP                                   COMBINATORIAL ;
INPUT
PIN  6          READ                                    COMBINATORIAL ;
INPUT
PIN  7          VCC                                                   ;
INPUT
PIN  8          /DS                                     COMBINATORIAL ;
INPUT
PIN  9          AD_CS                                   COMBINATORIAL ;
INPUT
PIN  10         AD1_NAD2                                COMBINATORIAL ;
INPUT
PIN  11         ATOD1D0                                 COMBINATORIAL ;
INPUT
PIN  12         ATOD2D0                                 COMBINATORIAL ;
INPUT
PIN  13         ATOD1D1                                 COMBINATORIAL ;
INPUT
PIN  14         ATOD2D1                                 COMBINATORIAL ;
INPUT
PIN  15         D17                                     COMBINATORIAL ;
OUTPUT
PIN  16         D16                                     COMBINATORIAL ;
OUTPUT
PIN  17         DSACK1                                  COMBINATORIAL ;
OUTPUT
PIN  18         CK8M                                    REGISTERED    ;
OUTPUT
PIN  19         /ACW                                    COMBINATORIAL ;
OUTPUT
PIN  20         /ACR                                    COMBINATORIAL ;
OUTPUT
PIN  21         GND                                                   ;
INPUT
PIN  22         CK128K                                  REGISTERED    ;
OUTPUT
PIN  23         CK256K                                  REGISTERED    ;
OUTPUT
PIN  24         CK512K                                  REGISTERED    ;
OUTPUT
PIN  25         CK1M                                    REGISTERED    ;
OUTPUT
PIN  26         CK2M                                    REGISTERED    ;
OUTPUT
PIN  27         CK4M                                    REGISTERED    ;
OUTPUT
```

```
;---------------------------------- Boolean Equation Segment ------
EQUATIONS
CK8M = /CK8M
CK4M = /CK4M * CK8M + CK4M * /CK8M
CK2M = /CK2M * CK8M * CK4M + CK2M * /(CK8M * CK4M)
CK1M = /CK1M * CK8M * CK4M * CK2M + CK1M * /(CK8M * CK4M * CK2M)
CK512K = /CK512K * CK8M * CK4M * CK2M * CK1M +
         CK512K * /(CK8M * CK4M * CK2M * CK1M)
CK256K = /CK256K * CK8M * CK4M * CK2M * CK1M * CK512K +
         CK256K * /(CK8M * CK4M * CK2M * CK1M * CK512K)
CK128K = /CK128K * CK8M * CK4M * CK2M * CK1M * CK512K * CK256K +
         CK128K * /(CK8M * CK4M * CK2M * CK1M * CK512K * CK256K)

CK8M.CLKF = CLK16
CK4M.CLKF = CLK16
CK2M.CLKF = CLK16
CK1M.CLKF = CLK16
CK512K.CLKF = CLK16
CK256K.CLKF = CLK16
CK128K.CLKF = CLK16

ACW = DS * /READ * SEG1 * /CPUSP
ACR = DS * READ * SEG1 * /CPUSP

DSACK1 = AD_CS

D16 = AD_CS * AD1_NAD2 * ATOD1D0 + AD_CS * /AD1_NAD2 * ATOD2D0
D16.TRST = AD_CS
D17 = AD_CS * AD1_NAD2 * ATOD1D1 + AD_CS * /AD1_NAD2 * ATOD2D1
D17.TRST = AD_CS
```

Table 9.7 PAL U34 Equations

```
;PALASM Design Description
□

□
;---------------------------------- Declaration Segment ------------
□
TITLE    K processor A/D bus driver and timer interupt
□
PATTERN  1-a
□
CHIP  kpal_u34  PALCE26V12

;---------------------------------- PIN Declarations ---------------
PIN  1          KTIME                                COMBINATORIAL ;
INPUT
PIN  2          AD_CS                                COMBINATORIAL ;
INPUT
```

```
PIN  3          AD1_NAD2                                COMBINATORIAL ;
INPUT
PIN  4          ATOD1_D[2]                              COMBINATORIAL ;
INPUT
PIN  5          ATOD2_D[2]                              COMBINATORIAL ;
INPUT
PIN  6          ATOD1_D[3]                              COMBINATORIAL ;
INPUT
PIN  7          VCC                                                   ;
INPUT
PIN  8          ATOD2_D[3]                              COMBINATORIAL ;
INPUT
PIN  9          ATOD1_D[4]                              COMBINATORIAL ;
INPUT
PIN  10         ATOD2_D[4]                              COMBINATORIAL ;
INPUT
PIN  11         ATOD1_D[5]                              COMBINATORIAL ;
INPUT
PIN  12         ATOD2_D[5]                              COMBINATORIAL ;
INPUT
PIN  13         ATOD1_D[6]                              COMBINATORIAL ;
INPUT
PIN  14         ATOD2_D[6]                              COMBINATORIAL ;
INPUT
PIN  15..20,22  D[24..18]                               COMBINATORIAL ;


OUTPUT
PIN  21         GND                                                   ;
INPUT
PIN  23         ATOD2_D[8]                              COMBINATORIAL ;
INPUT
PIN  24         ATOD1_D[8]                              COMBINATORIAL ;
INPUT
PIN  25         ATOD2_D[7]                              COMBINATORIAL ;
INPUT
PIN  26         ATOD1_D[7]                              COMBINATORIAL ;
INPUT
PIN  27         /IRQ3                                   REGISTERED    ;
OUTPUT
PIN  28         KPC1                                    COMBINATORIAL ;
INPUT
NODE  1         GLOBAL

;---------------------------------- Boolean Equation Segment ------
EQUATIONS
D[18..24] = AD_CS * AD1_NAD2 * ATOD1_D[2..8] +
            AD_CS * /AD1_NAD2 * ATOD2_D[2..8]
D[18..24].TRST = AD_CS

IRQ3 = VCC
IRQ3.CLKF = KTIME
GLOBAL.RSTF = KPC1
GLOBAL.SETF = GND
```

Table 9.8 PAL U301 Equations

```
□
;PALASM Design Description
□
;
□
; Revision History
□
;
□
;
□
;              Addresses SMCS1 = %X4003C000  SMDSACK0 Generated
;                        SMCS2 = %X4003C400  NO SMDSACK Generated
;                        KAC21 = %X4003D000  SMDSACK0 Generated
;                        KAC22 = %X4003D400  SMDSACK0 Generated


;---------------------------------- Declaration Segment ------------
TITLE    SEGMENT DECODER PAL
PATTERN  A
REVISION 1.0

CHIP   KPALU301  PAL22V10

;---------------------------------- PIN Declarations ---------------
PIN   1         RNW                  COMBINATORIAL             ; INPUT
PIN   2         /SMCSIN              COMBINATORIAL             ; INPUT
PIN   3         WAIT2                COMBINATORIAL             ; INPUT
PIN   9         A10                  COMBINATORIAL             ; INPUT
PIN  10         A11                  COMBINATORIAL             ; INPUT
PIN  11         A12                  COMBINATORIAL             ; INPUT
PIN  12         GND                                            ; INPUT
PIN  13         A13                  COMBINATORIAL             ; INPUT
PIN  14         /SMCS2               COMBINATORIAL             ; OUTPUT
PIN  15         /SMCS1               COMBINATORIAL             ; OUTPUT
PIN  16         /KAC22               COMBINATORIAL             ; OUTPUT
PIN  17         /KAC21               COMBINATORIAL             ; OUTPUT
PIN  18         /SMBUFFEN            COMBINATORIAL             ; OUTPUT
PIN  19         SMDSACK0             COMBINATORIAL             ; OUTPUT
PIN  21         SMDSACK1             COMBINATORIAL             ; OUTPUT
PIN  22         /SCAN                COMBINATORIAL             ; OUTPUT
PIN  23         RD                   COMBINATORIAL              ; OUTPUT
PIN  24         VCC                                            ; INPUT

;OUTPUT EQUATIONS----------------------
EQUATIONS

RD       =  RNW * SMCSIN * /A13 * /A12 * /A11 * /A10
SCAN     = /RNW * SMCSIN * /A13 * /A12 * /A11 * /A10

SMCS1 = SMCSIN * /A13 * /A12 * /A11 * /A10
```

```
SMCS2 = SMCSIN * /A13 * /A12 * /A11 * A10
KAC21 = SMCSIN * /A13 * A12 * /A11 * /A10
KAC22 = SMCSIN * /A13 * A12 * /A11 * A10

SMBUFFEN = SMCS1 + SMCS2 + SCAN + RD

; Of the following two dsacks, only one should be enabled.
; If using the smart A/D board, enable SMDSACK0
; If using the 16-Bit A/D, enable SMDSACK1
; Disable a DSACK by setting it equal to GND
; (Either comment out the two middle or two outer equations)
SMDSACK0 = WAIT2 * (SMCS1 + KAC21 + KAC22)
; SMDSACK0 = GND
; SMDSACK1 = SCAN * WAIT2
SMDSACK1 = GND
```

Detail View CPU 68020

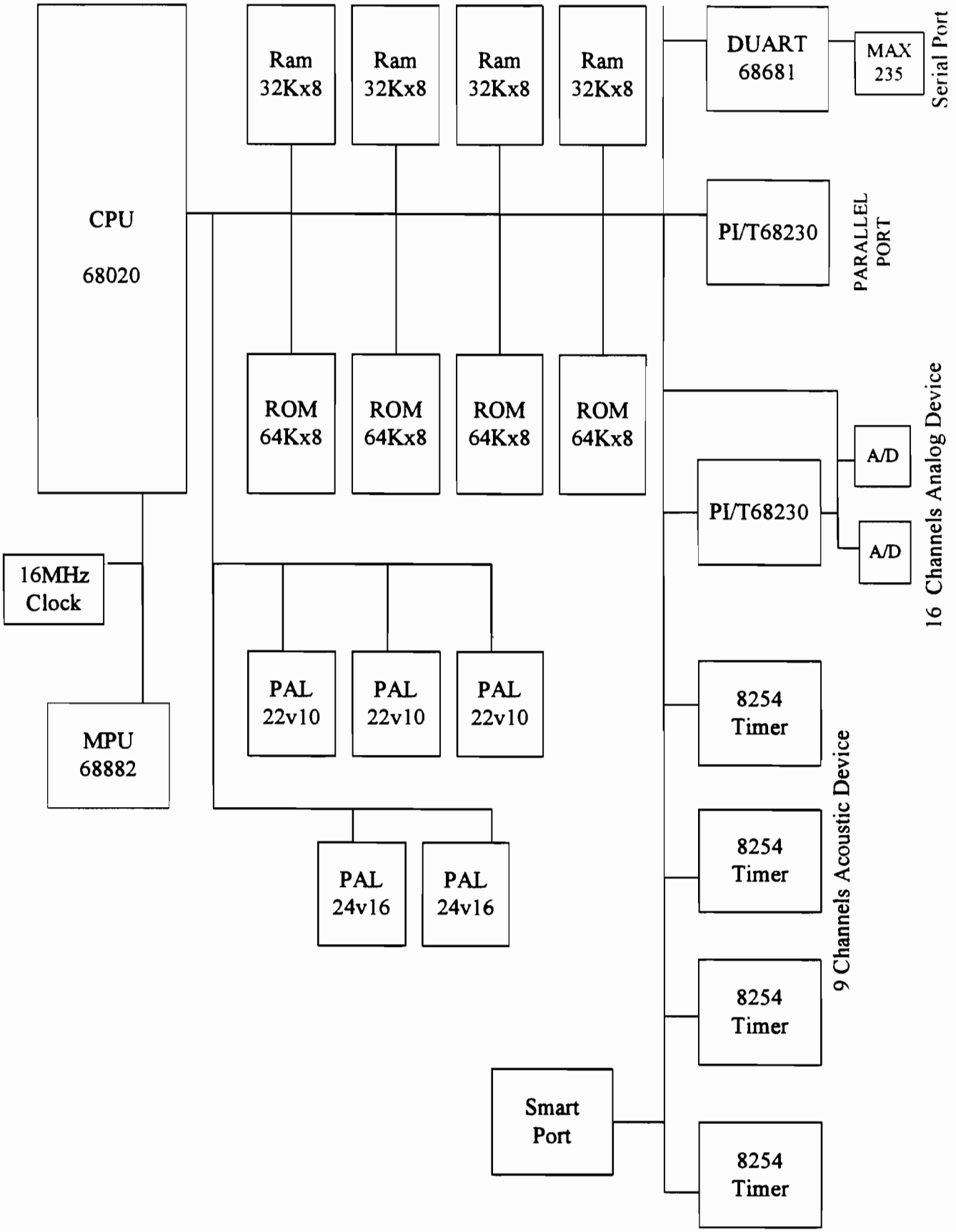


Figure 9.1 Detail View-CPU 68020

Overall View CPU 68020

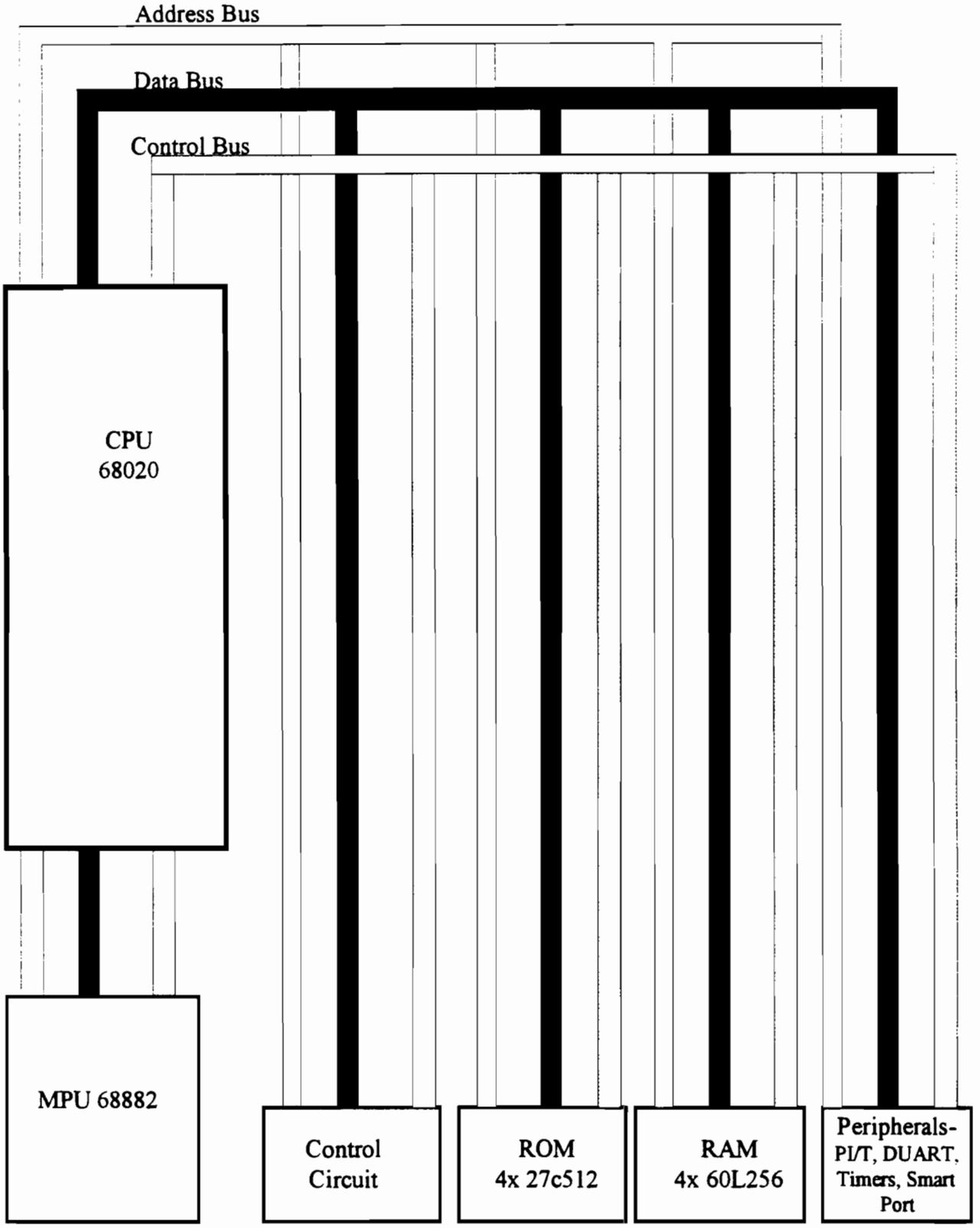


Figure 9.2 Over all View-CPU 68020

K BOARD LAYOUT

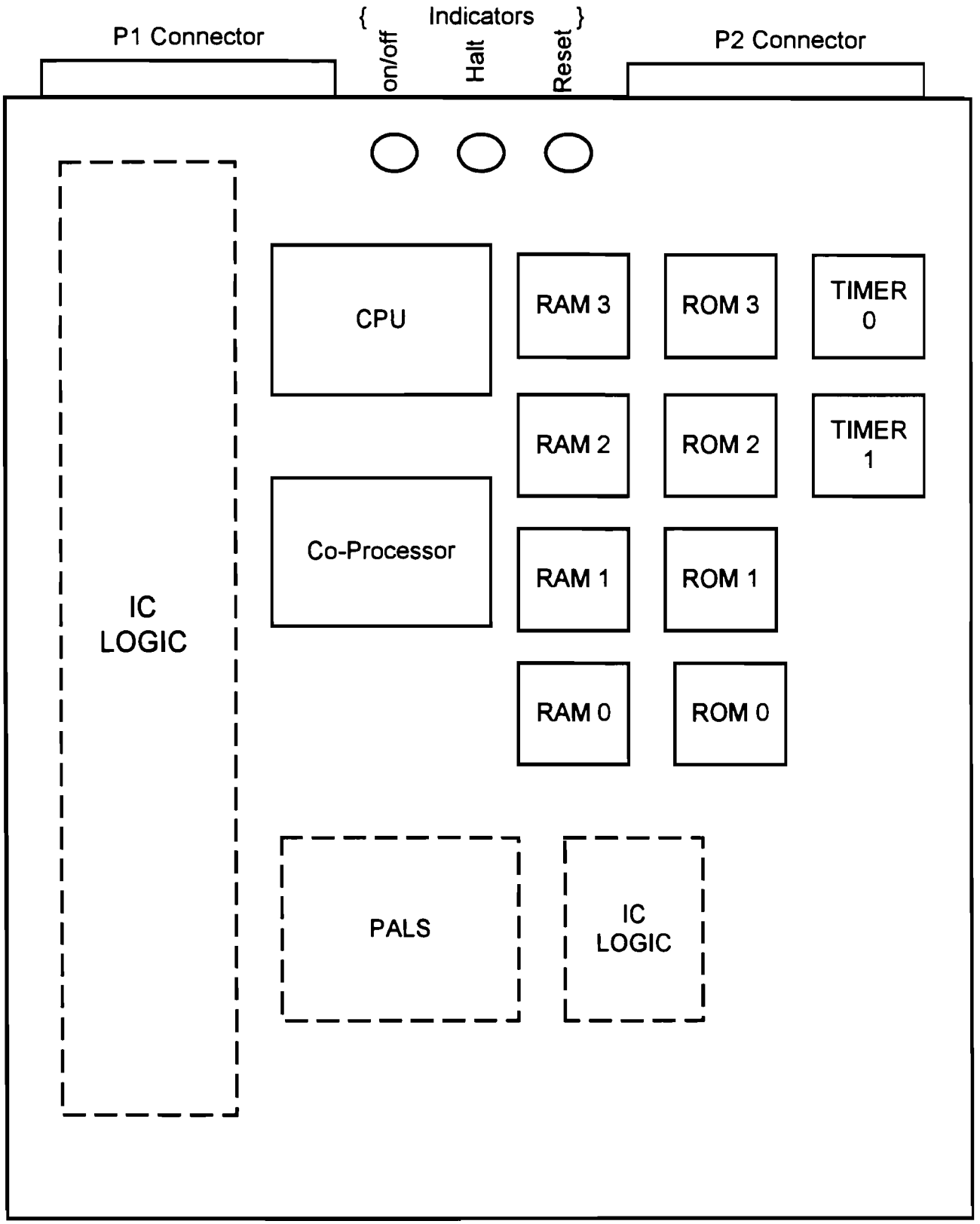


Figure 9.3 K Board Layout

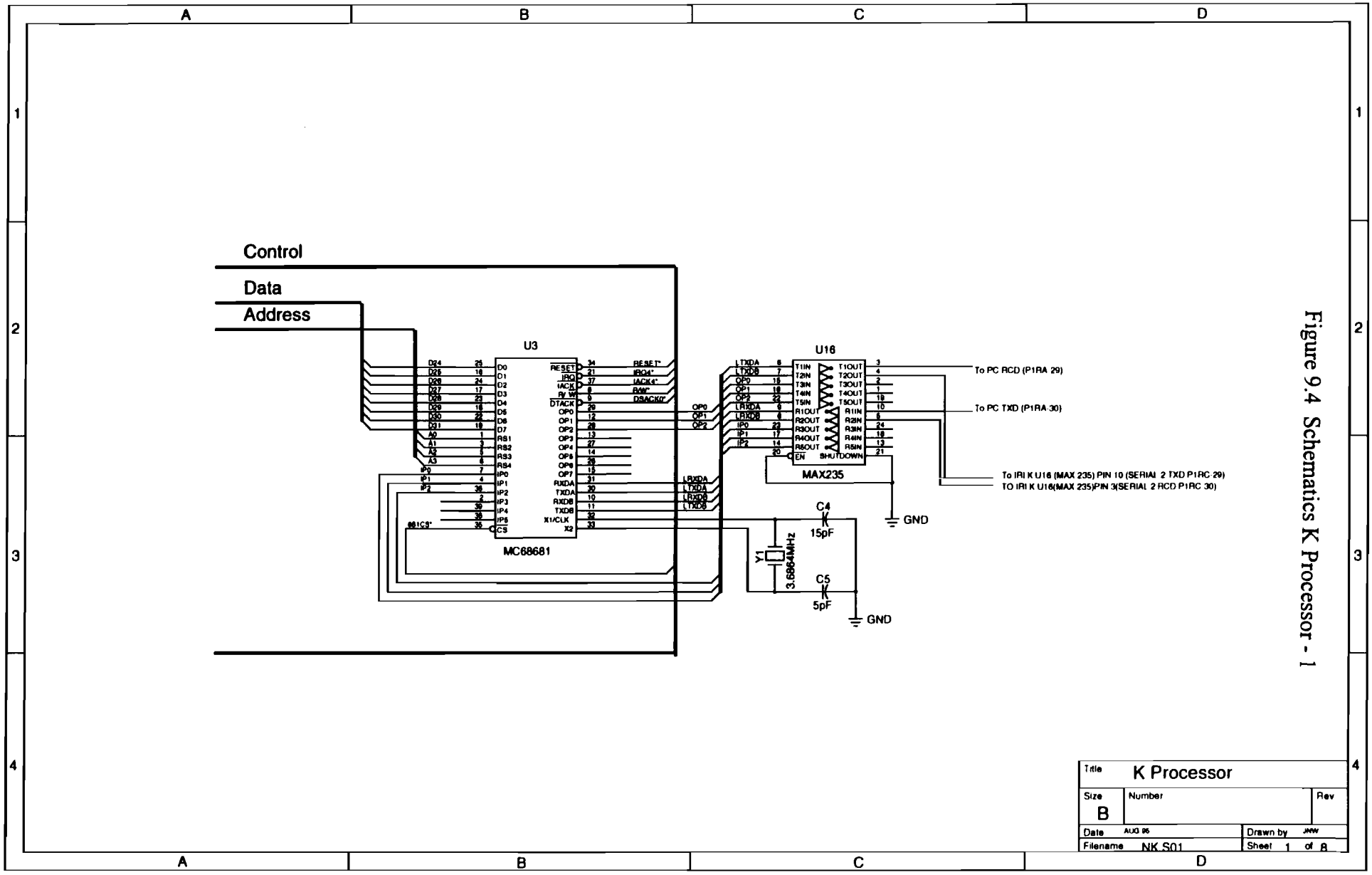


Figure 9.4 Schematics K Processor - 1

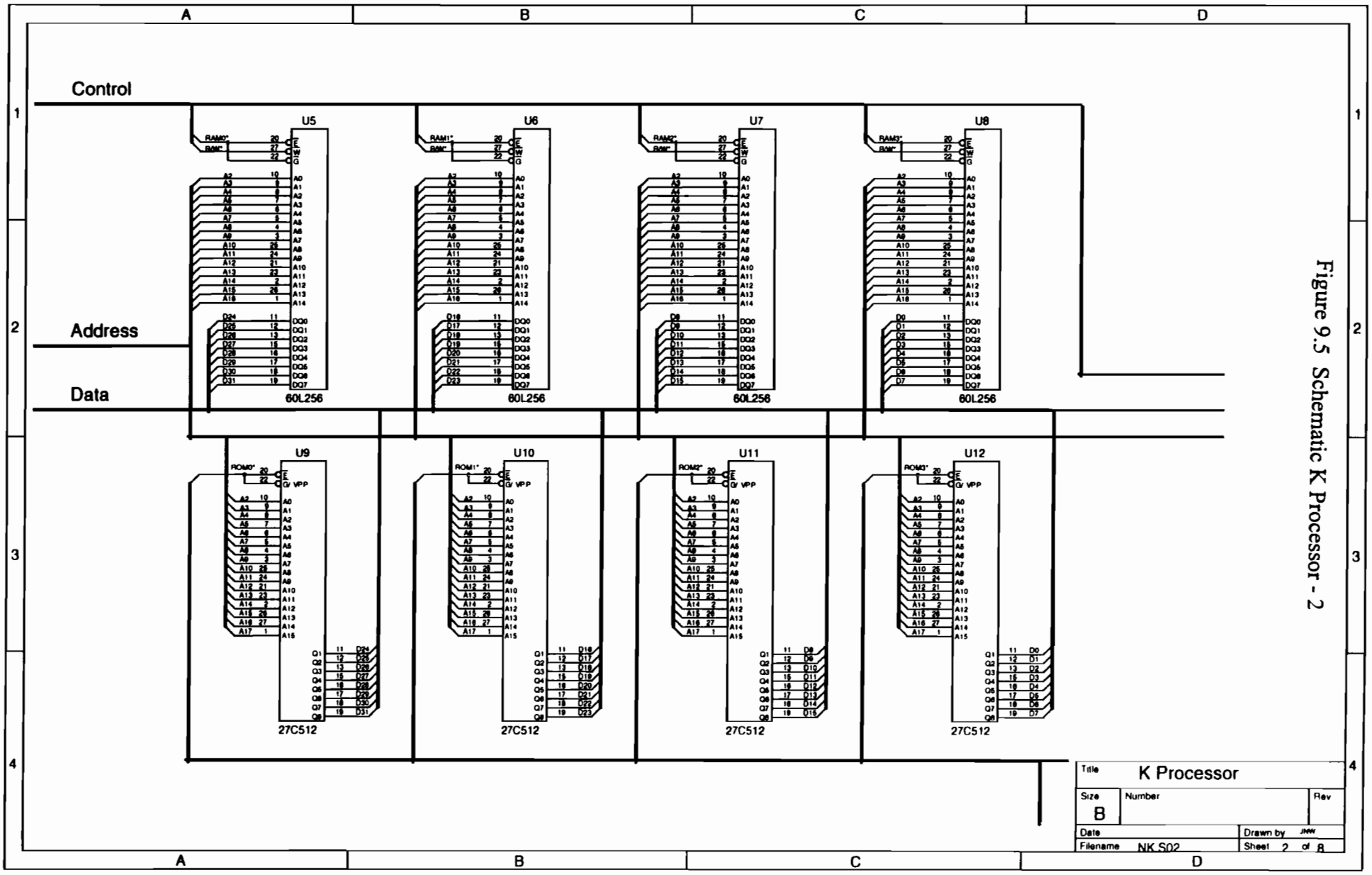


Figure 9.5 Schematic K Processor - 2

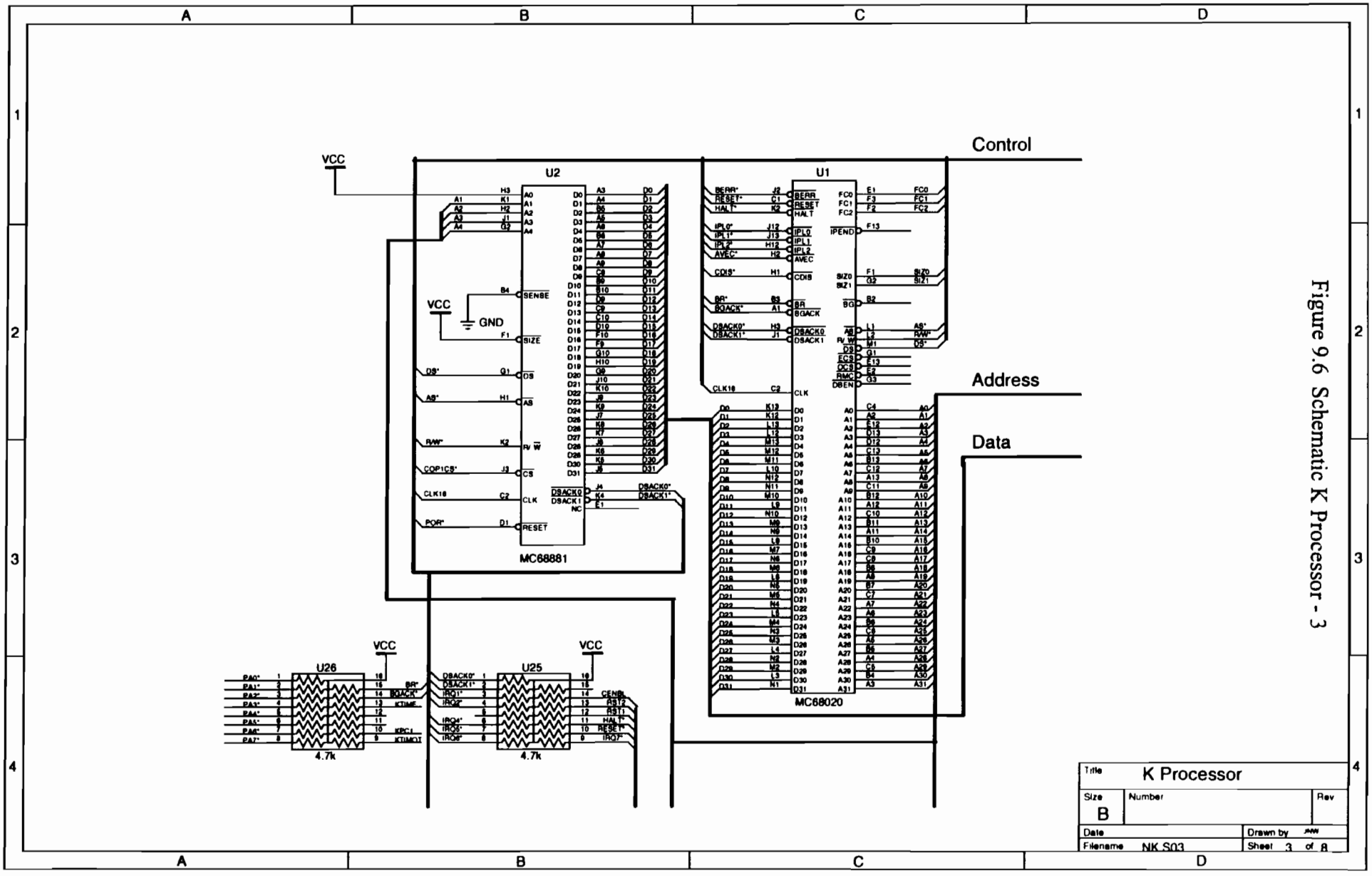


Figure 9.6 Schematic K Processor - 3

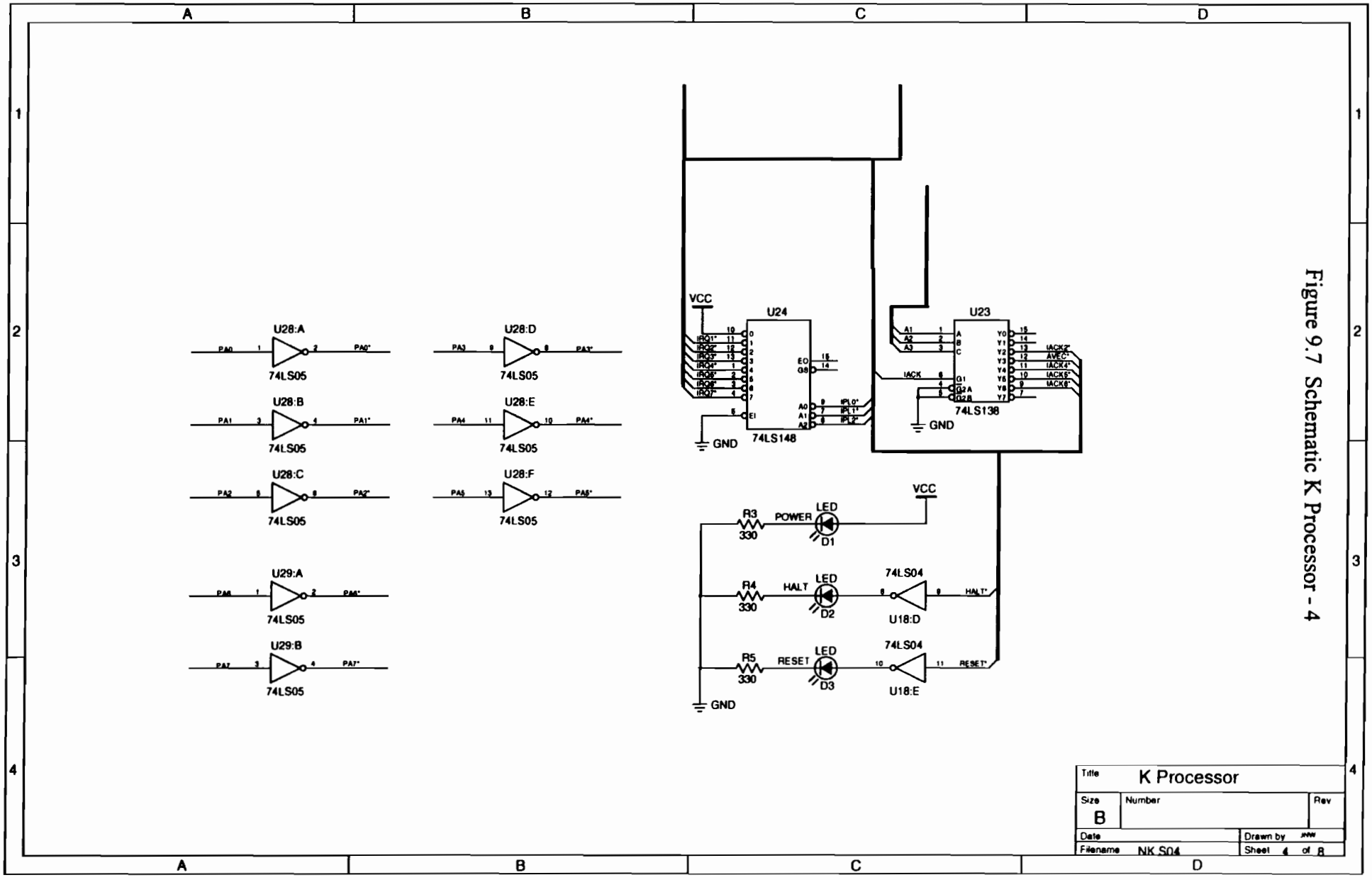
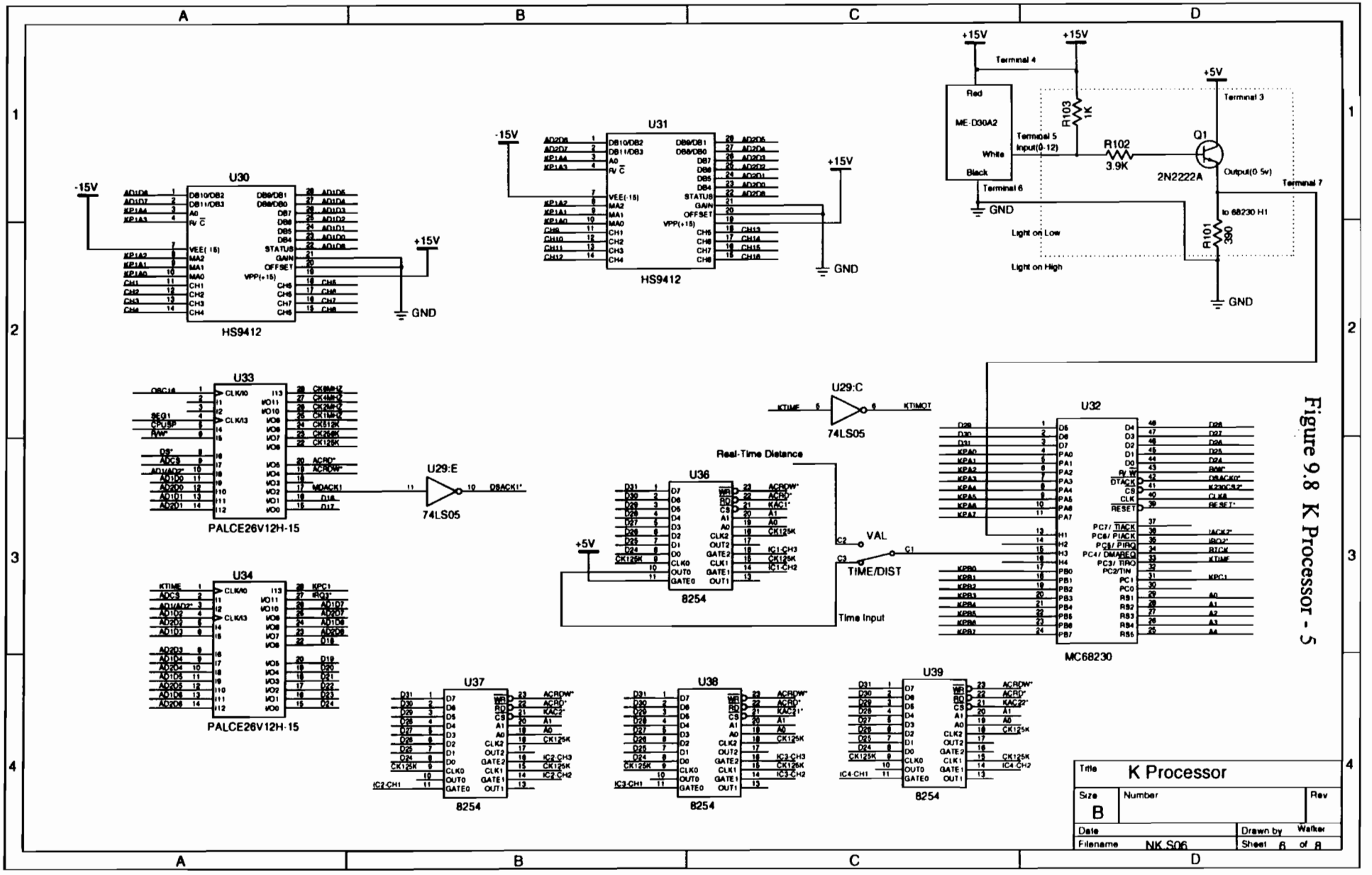


Figure 9.7 Schematic K Processor - 4


A
B
C
D
1
2
3
4
U30
HS9412
-15V
+15V
GND
U31
HS9412
-15V
+15V
GND
+15V
+15V
+5V
Terminal 4
Terminal 3
Red
ME-D30A2
White
Black
Terminal 5
Input(0-12)
Terminal 6
GND
R103
1K
R102
3.9K
Q1
2N2222A
Output(0 5v)
Terminal 7
to 68230 H1
R101
390
Light on Low
Light on High
GND
U33
PALCE26V12H-15
U29:E
74LS05
U29:C
74LS05
U34
PALCE26V12H-15
Real-Time Distance
U36
8254
+5V
VAL
TIME/DIST
TIme Input
U32
MC68230
U37
8254
U38
8254
U39
8254
Figure 9.8 K Processor - 5
Title
K Processor
Size
B
Number
Rev
Date
Drawn by
Walker
Filename
NK.S06
Sheet 6 of 8
92

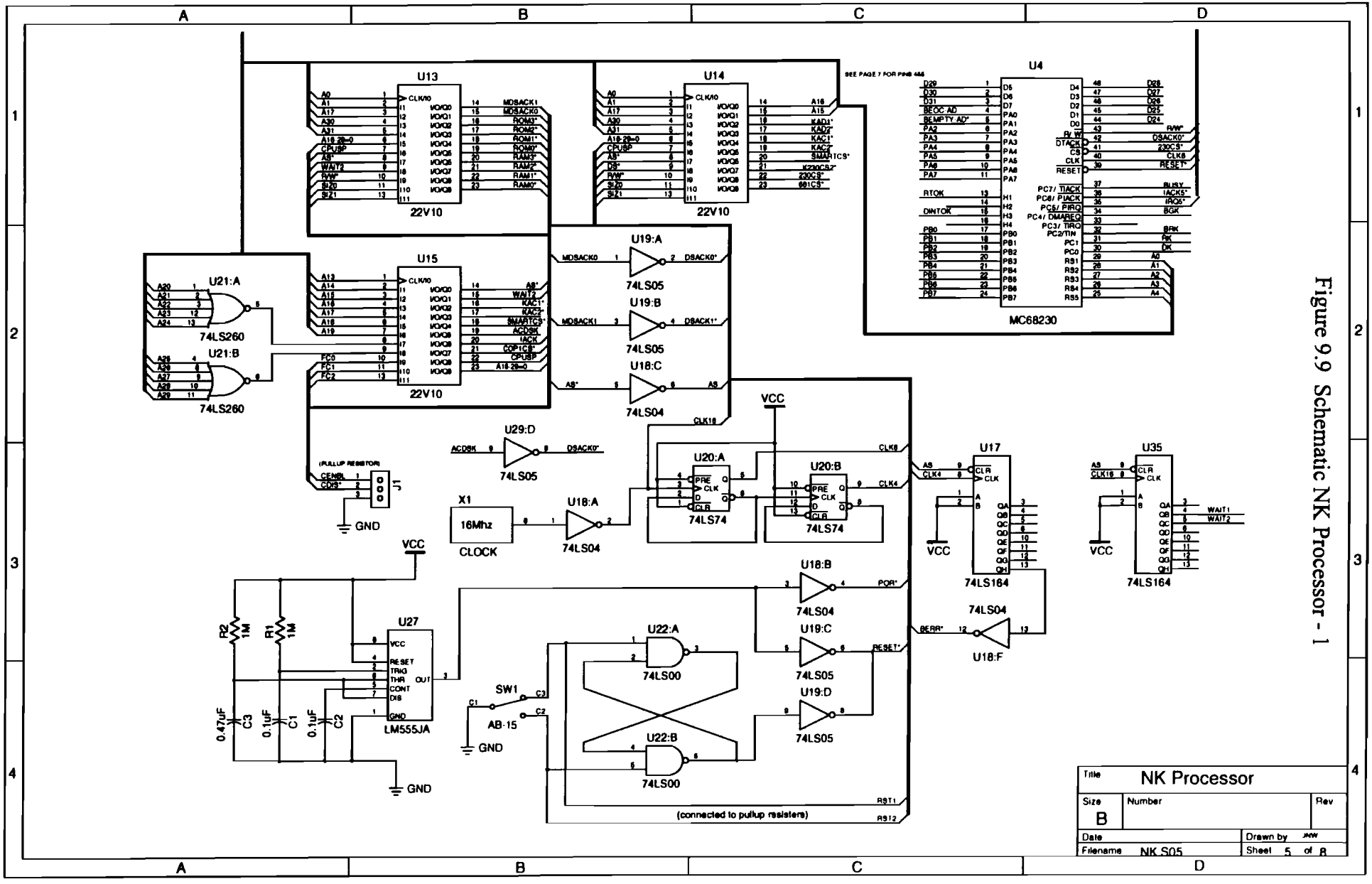


Figure 9.9 Schematic NK Processor - 1

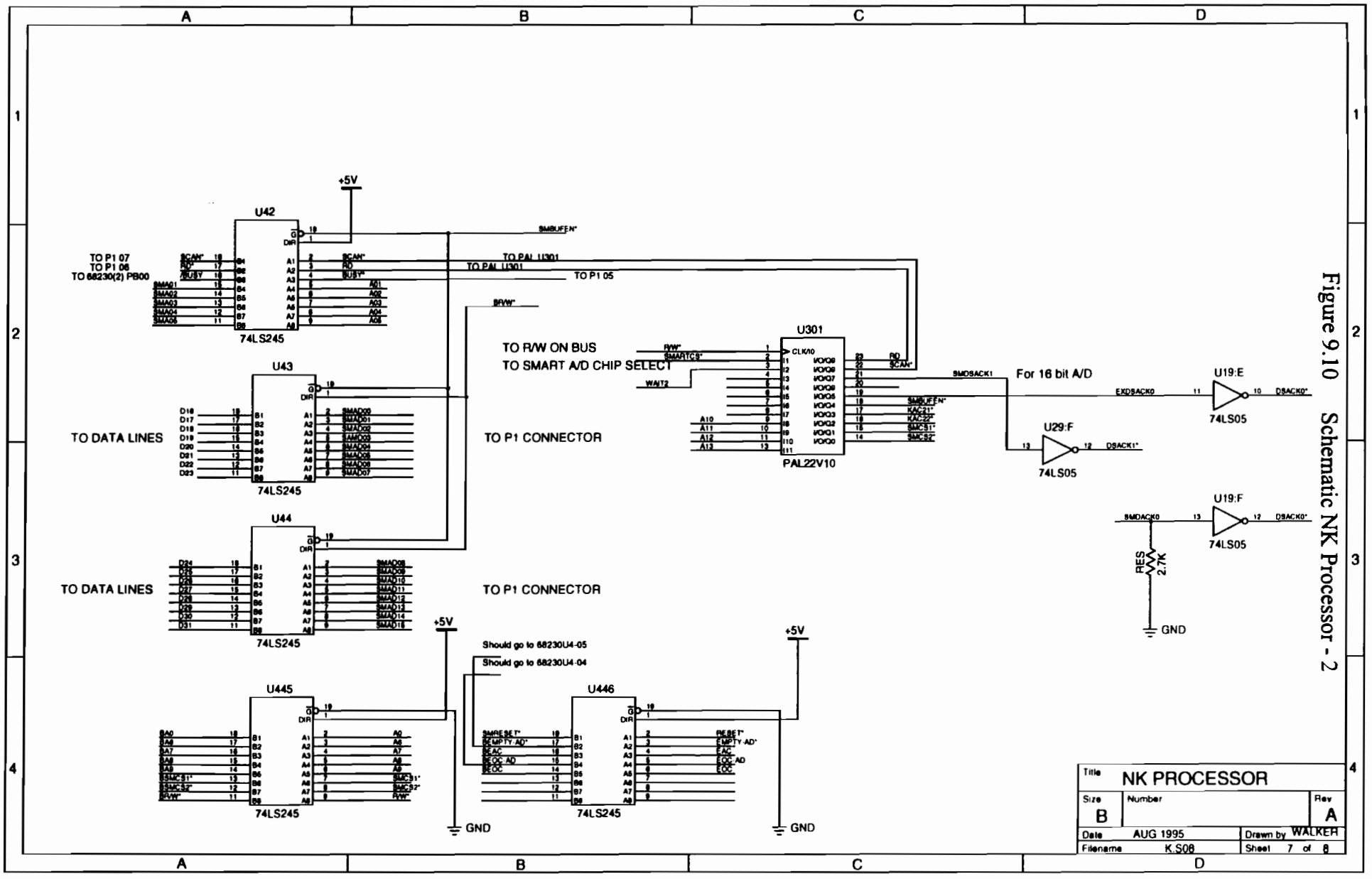
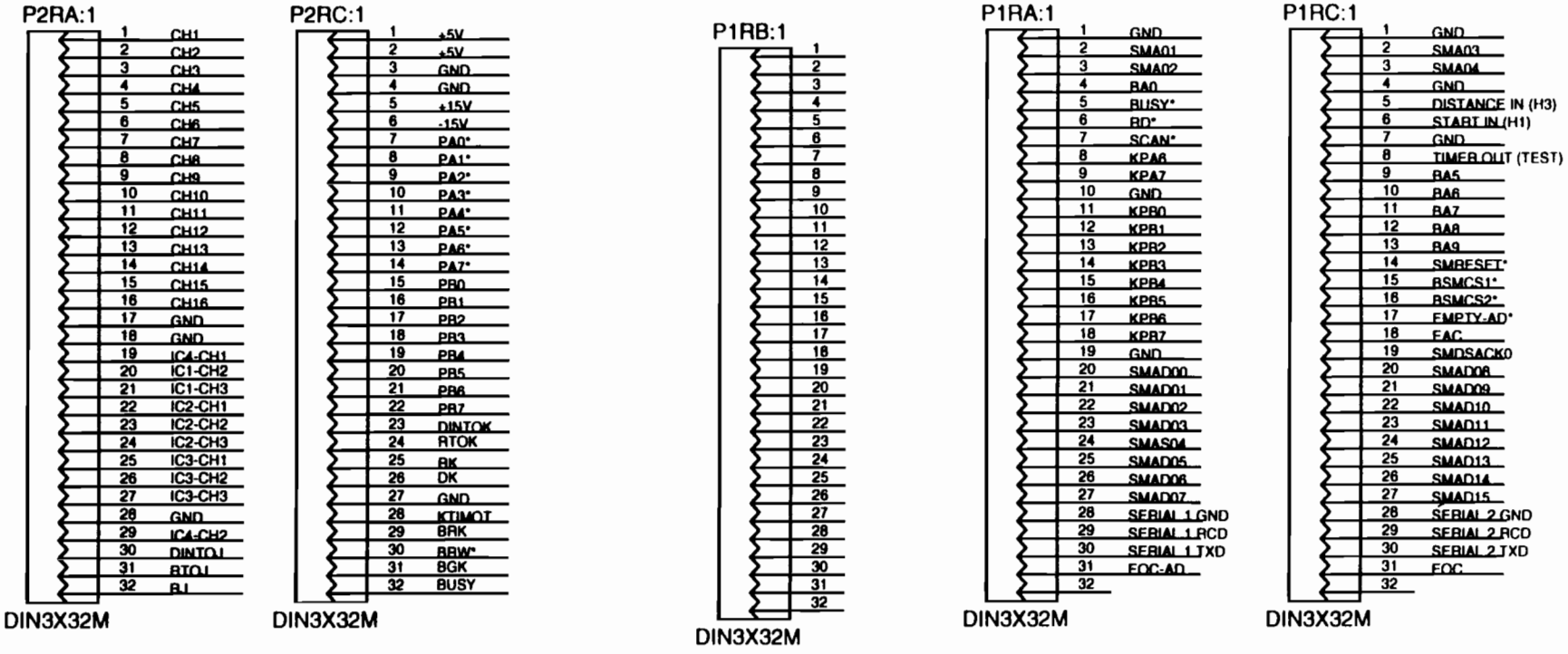


Figure 9.10 Schematic NK Processor - 2



NOTE: EOC-AD* WAS MOVED FROM
P1RC-32 TO P1RA-31

P1RB 16 NO RESET
17 CENTER RESET
18 NC RESET

Title	NK Processor	
Size A	Number	Rev
Date		Drawn by Walker
Filename NK S08		Sheet 8 of 8

Figure 9.11 Schematic NK Processor – 3

APPENDIX A

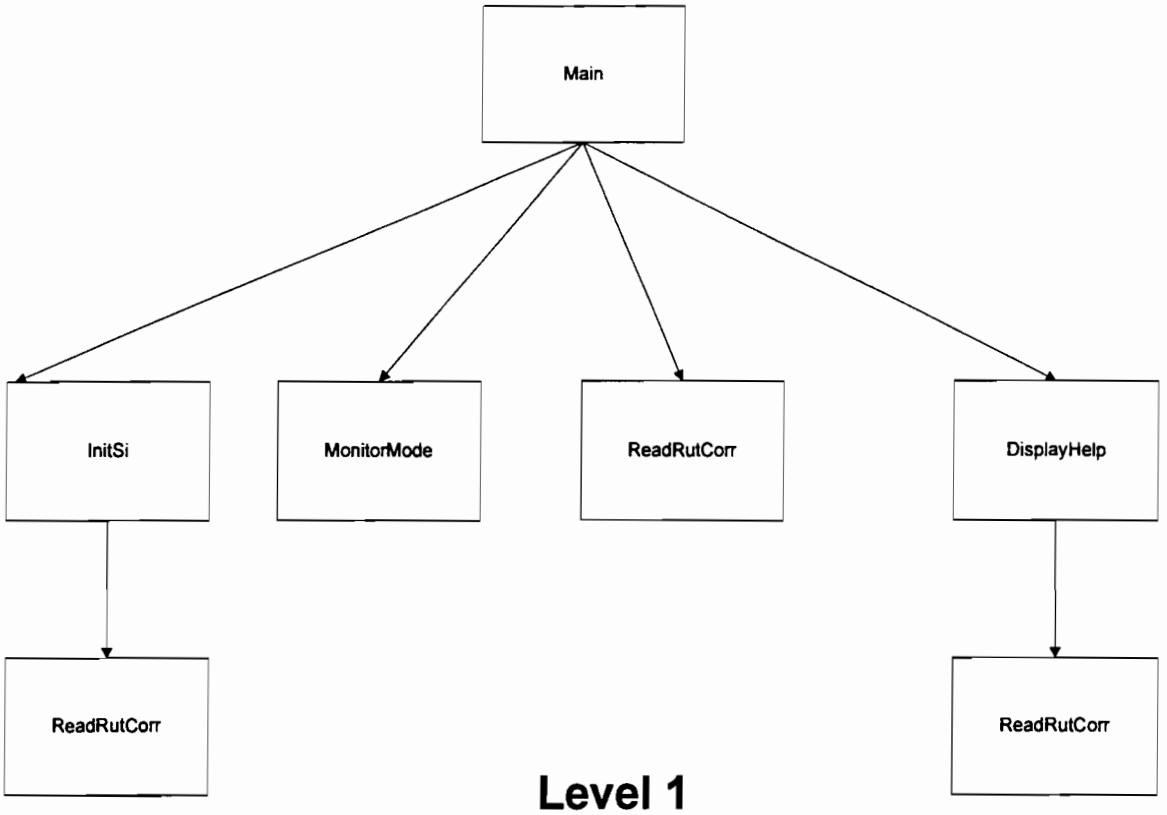
APPENDIX A

TALK INTRODUCTION

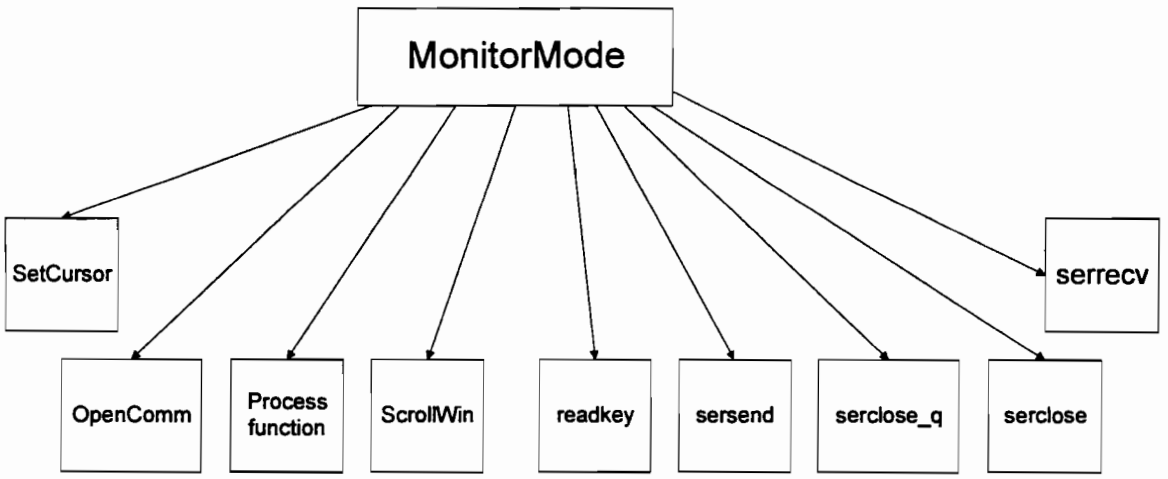
This documentation is for the program talk70.exe.
Talk70.exe uses the following files:
TALK70.H - shared include file
TALK70.C - main program, mode 1, mode 2, mode 4, and general purpose functions
BT.C - serial port communication and display functions
Mode3.C - mode 3 rut/si
SD1.C - mode 5
MODE6.C - mode 6
MODE7.C - mode 7
RTRUT.C - real-time rut
SERQS.ASM - serial port functions, supporting 3 ports
SERIALS.ASM - serial port functions, supporting 2 ports

The make file .tall 70.mak is used for creating the executable talk70.exe.
This document gives the description of each function in each of the files.
This description of each function is called function header. Partial structure charts are drawn to explain which function calls which function.
Some flowcharts are drawn to ease the understanding of some of the harder functions. We have also explained some of the global variables and what role they play in the program. The program talk70.exe uses some files which it writes to and reads from. This is also taken care of.
A Warnier Orr diagram is drawn which gives us the over all picture of the whole program. This diagram is also provided on disk. This can be loaded into any ascii editor and you can follow the flow of the program. The Warnier Orr diagram is useful in providing the function hierarchy indicating which function is calling which function(s). This helps in identifying the leaf functions (the functions that do not call any other function) which can provide a starting point for documentation.

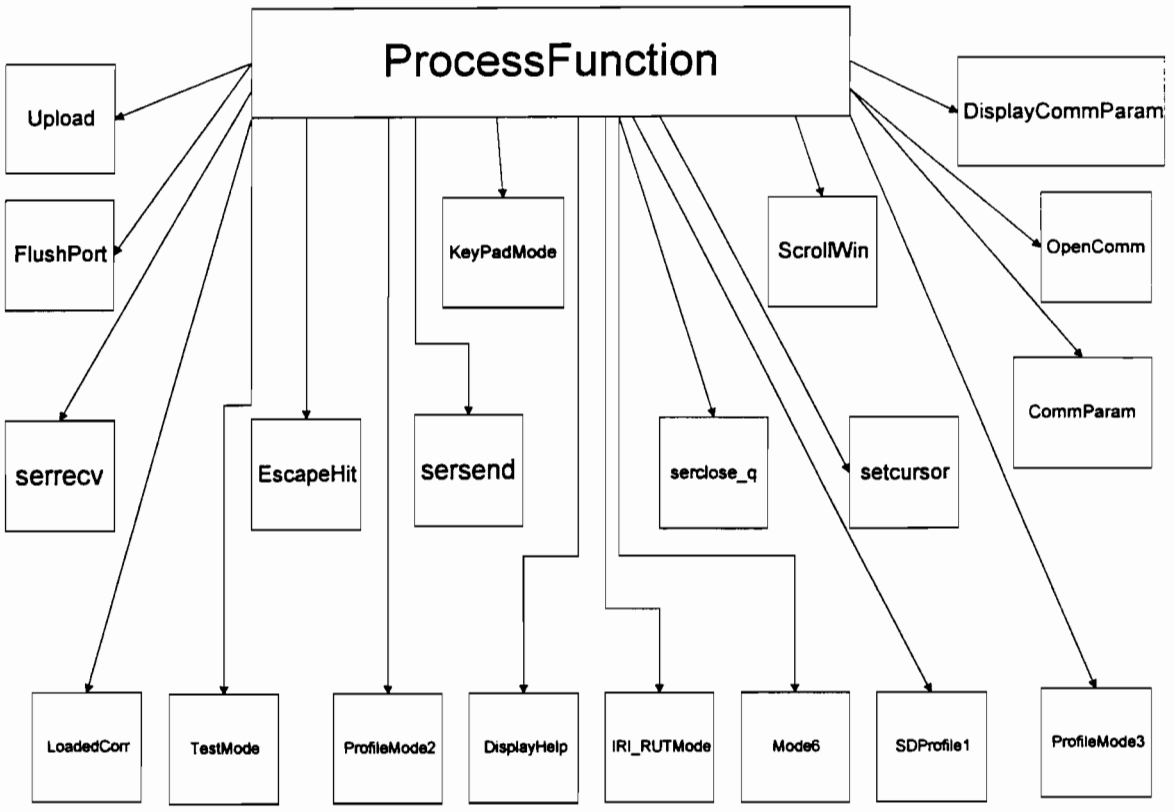
Structure Charts



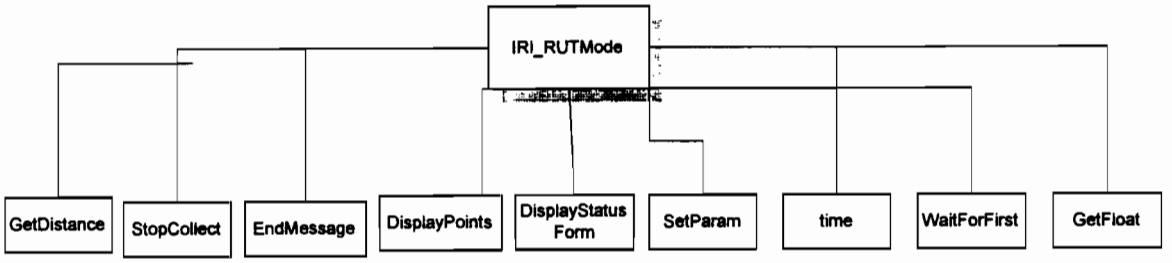
Level 1



Level 2

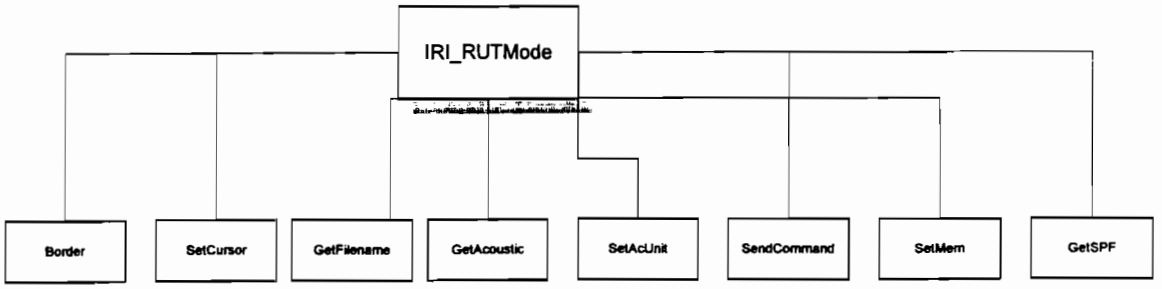


Level 3

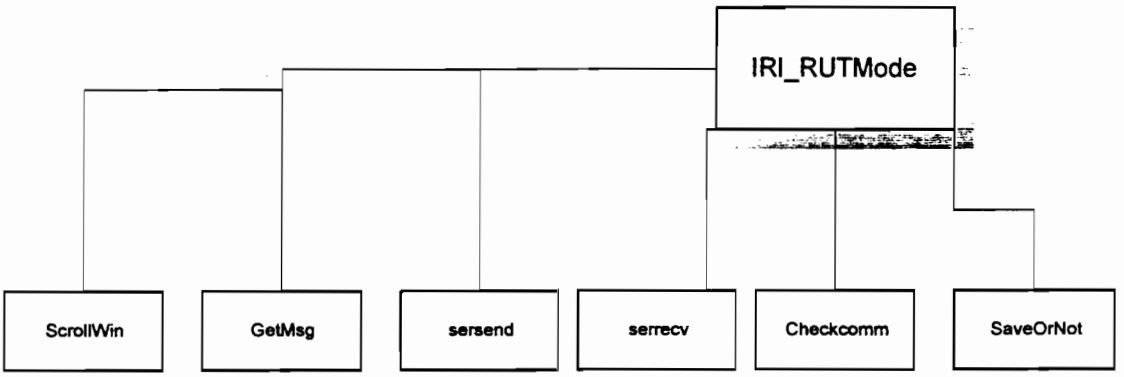


IRI_RUTMode

Level 4

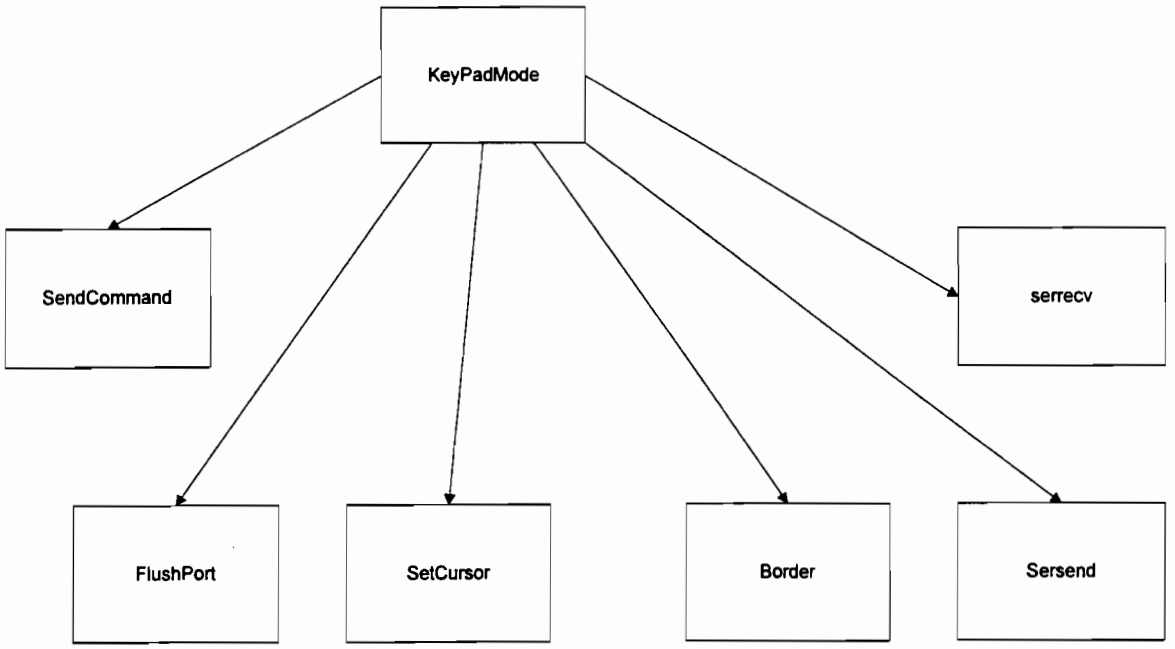


IRI_RUTMode Level 4

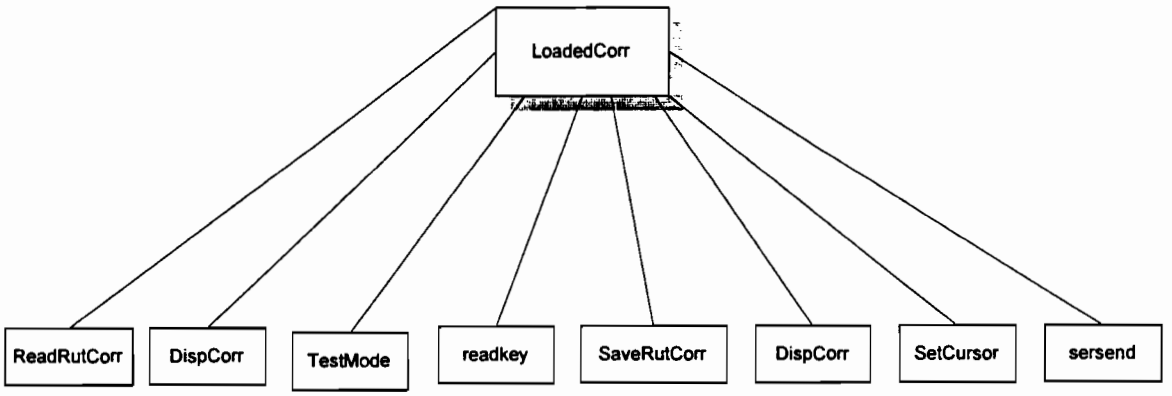


Level 4

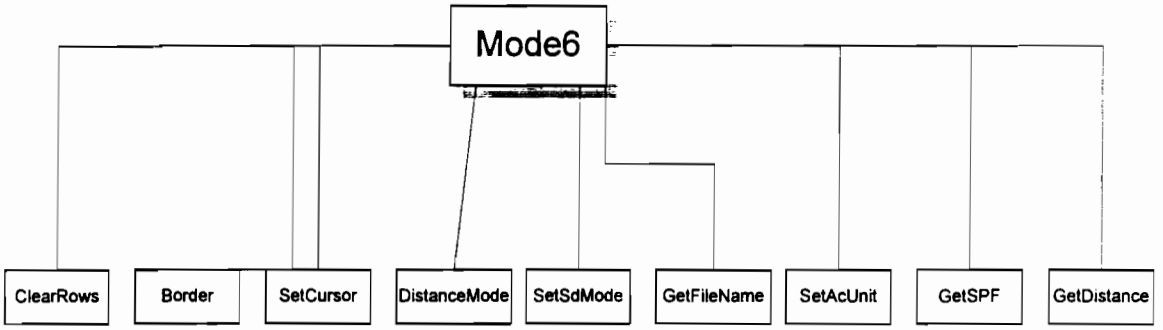
IRI_RUTMode



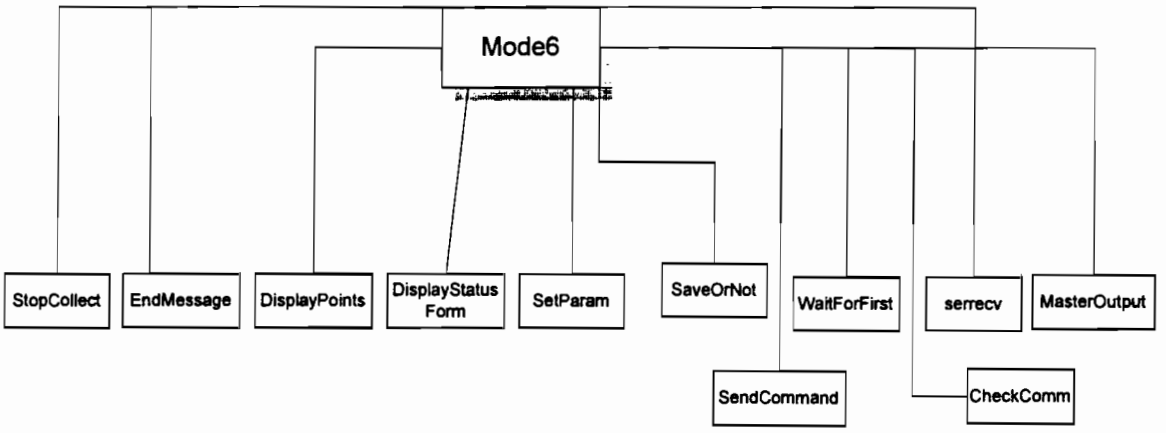
KeypadMode Level 4



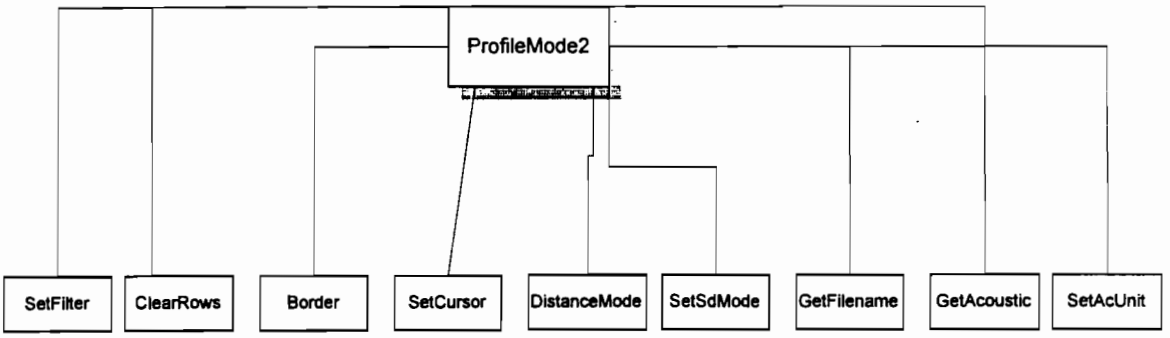
LoadedCorr Level 4



Mode6 Level 4

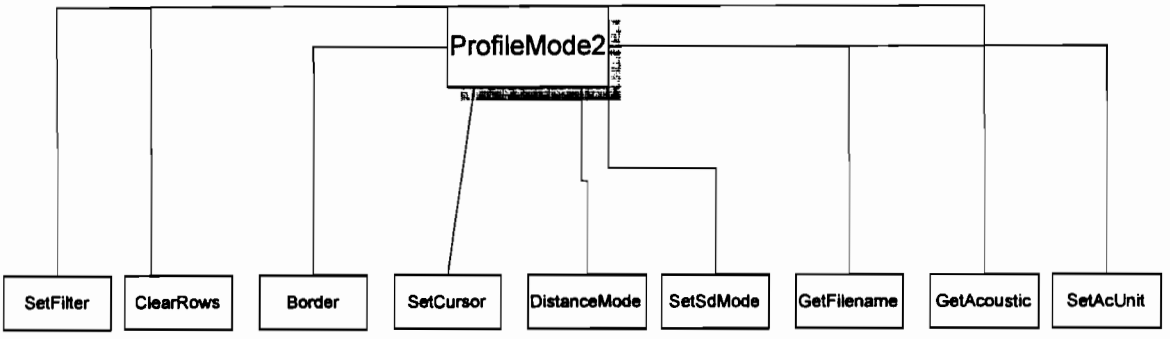


Mode6 Level 4



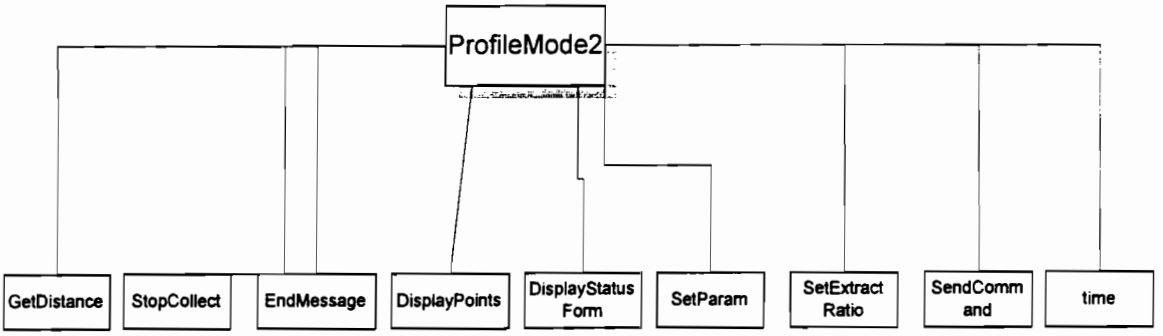
ProfileMode2

Level 4

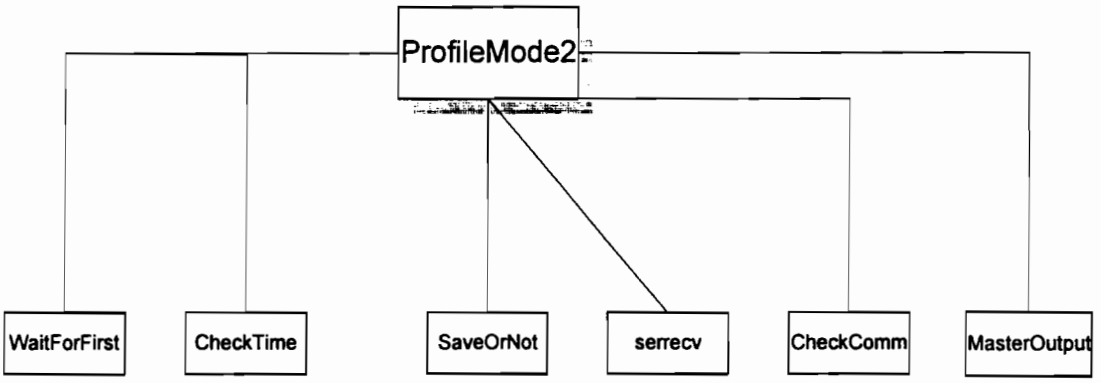


ProfileMode2

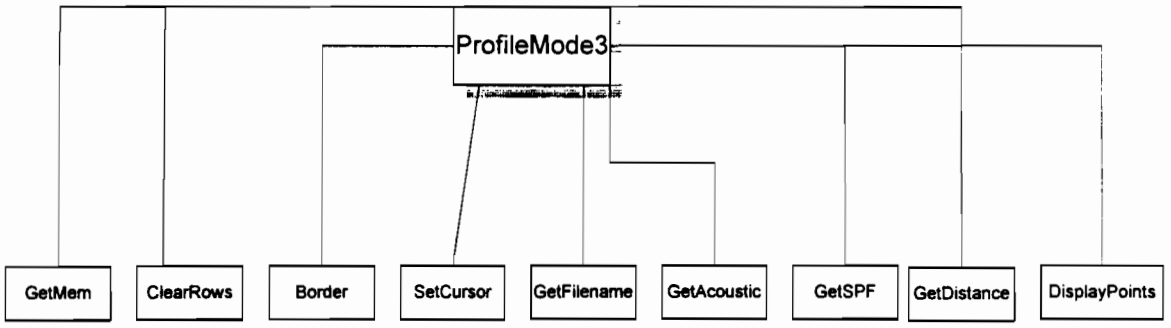
Level 4



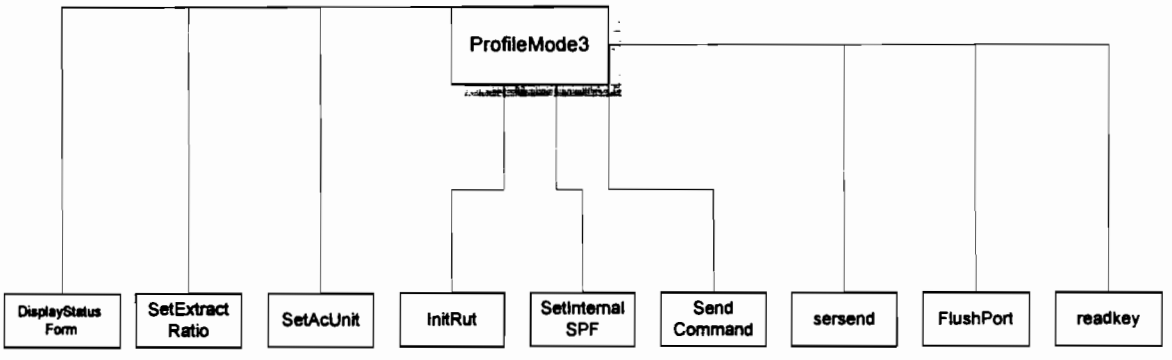
ProfileMode2 Level 4



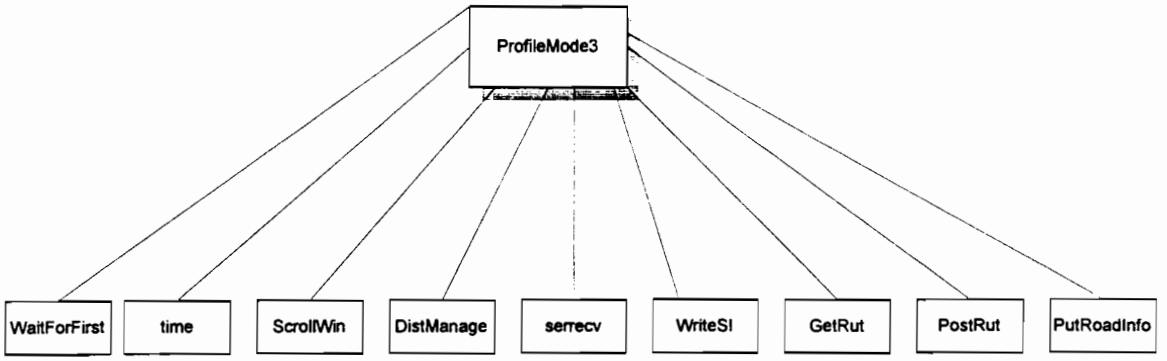
ProfileMode2 Level 4



ProfileMode3 Level 4

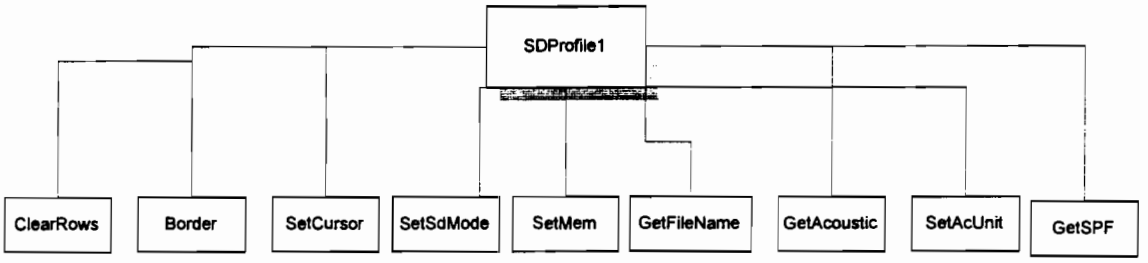


ProfileMode3 Level 4

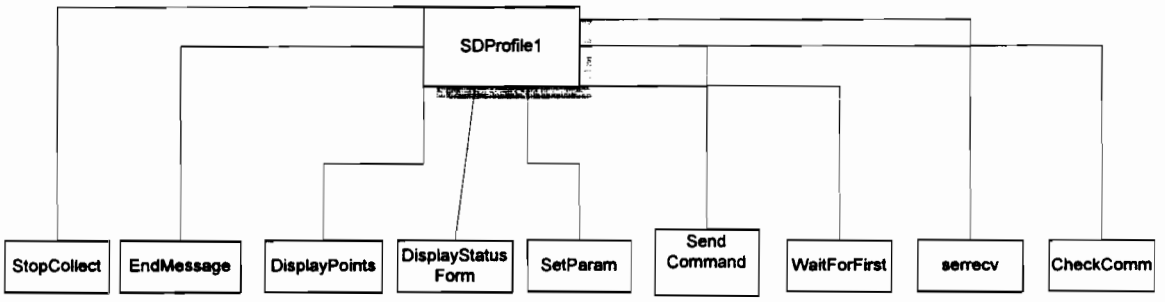


ProfileMode3 **Level 4**

Level 4

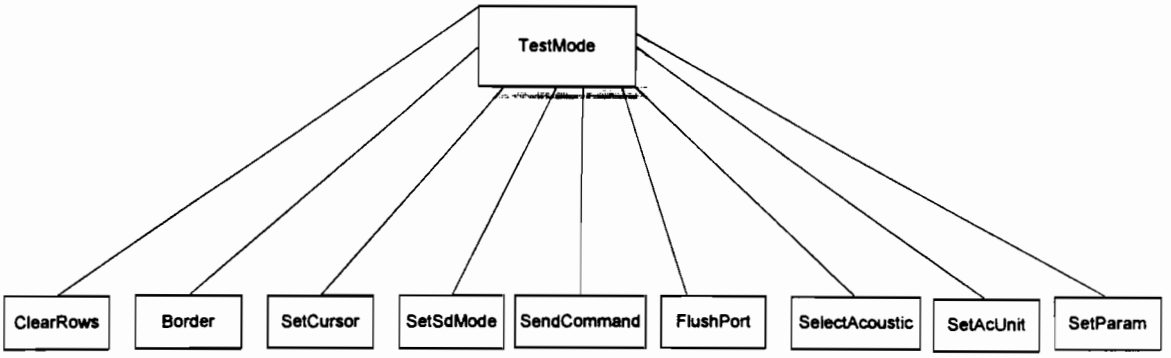


SDProfile1

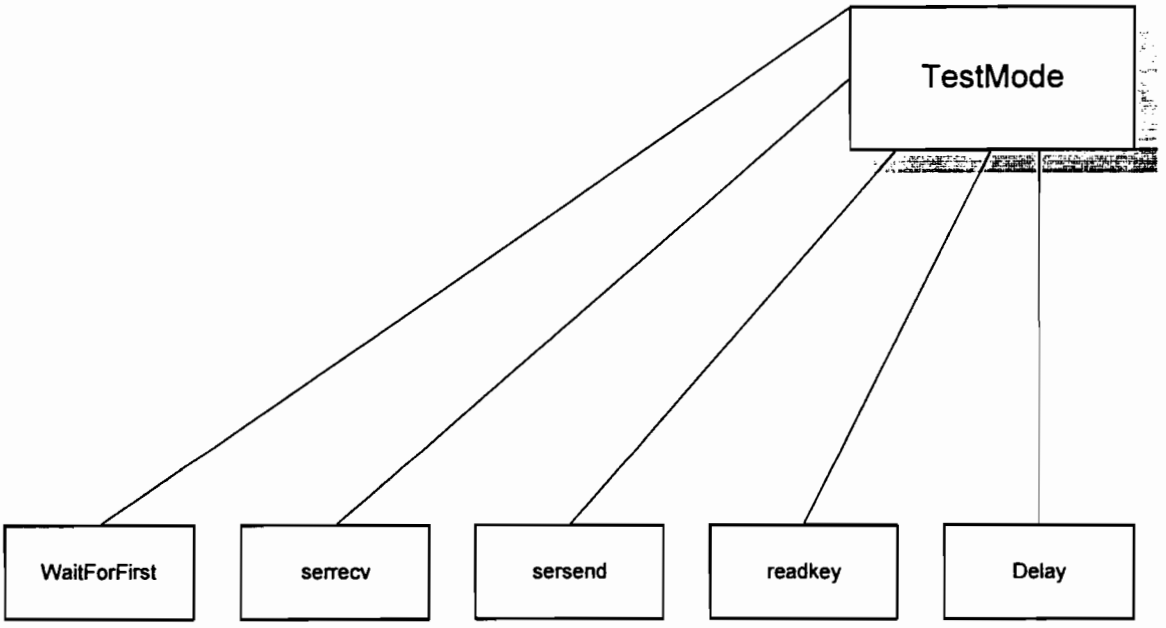


SDProfie1

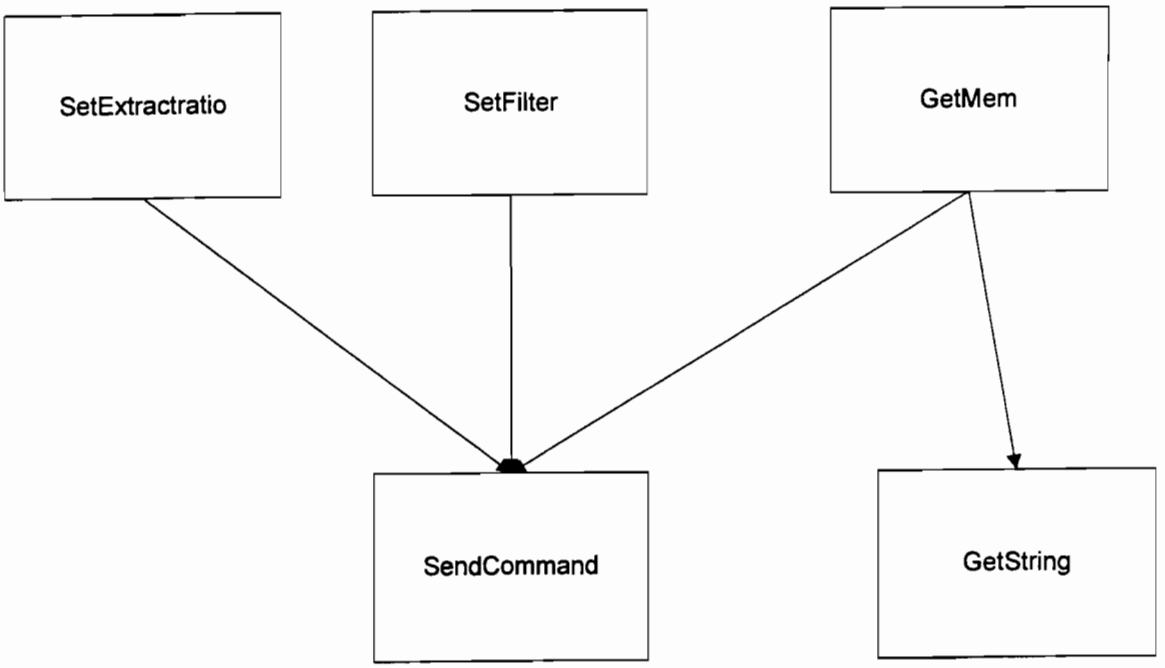
Level 4



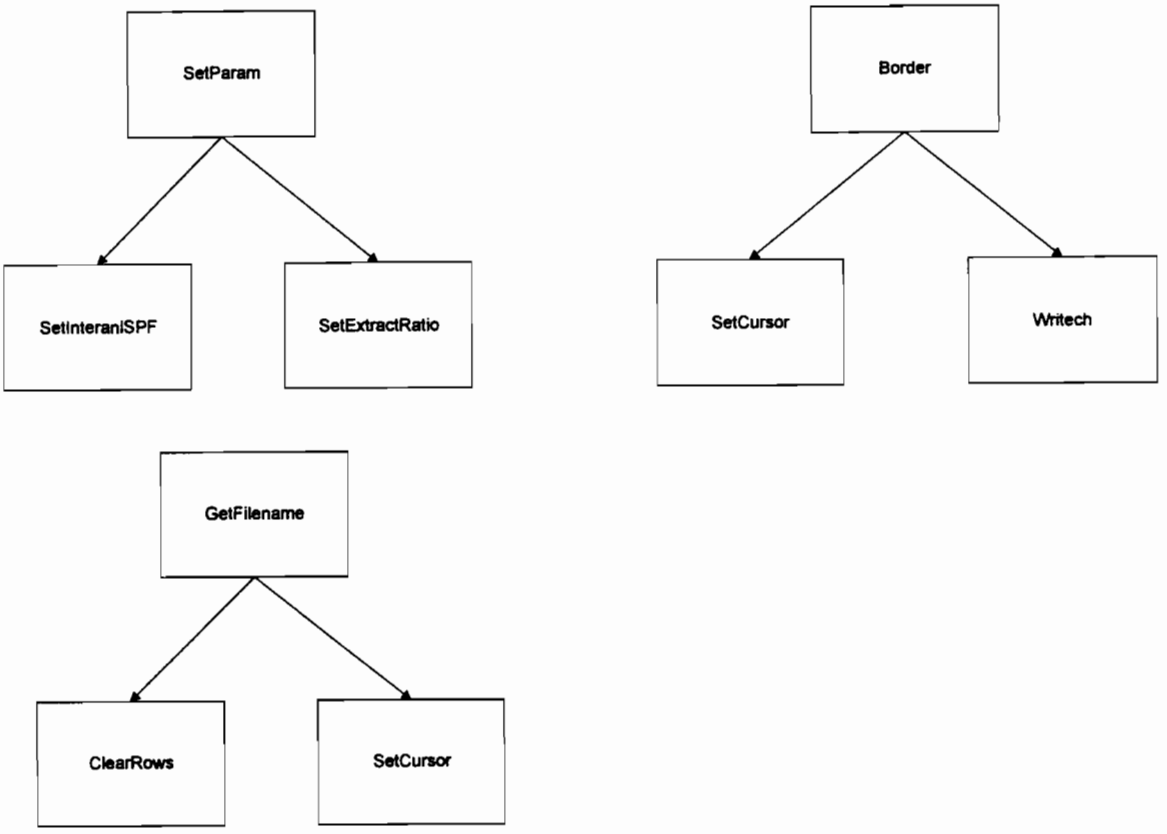
TestMode **Level 4**



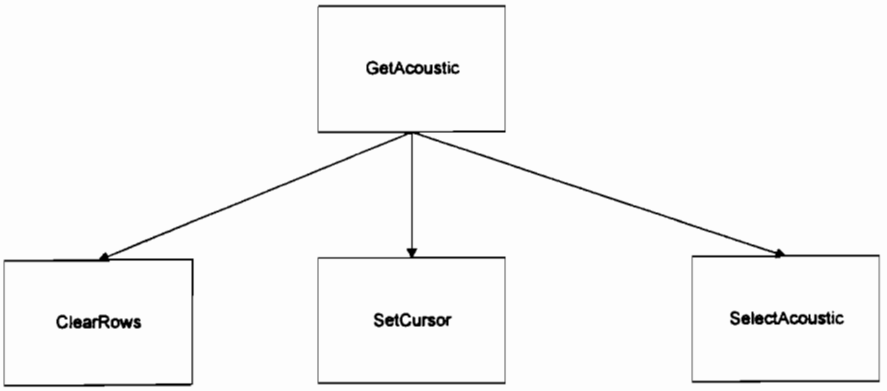
TestMode Level 4

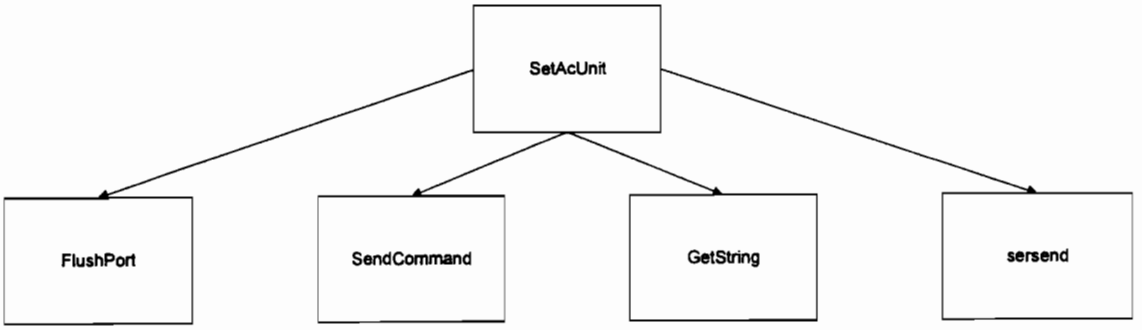


Level 5

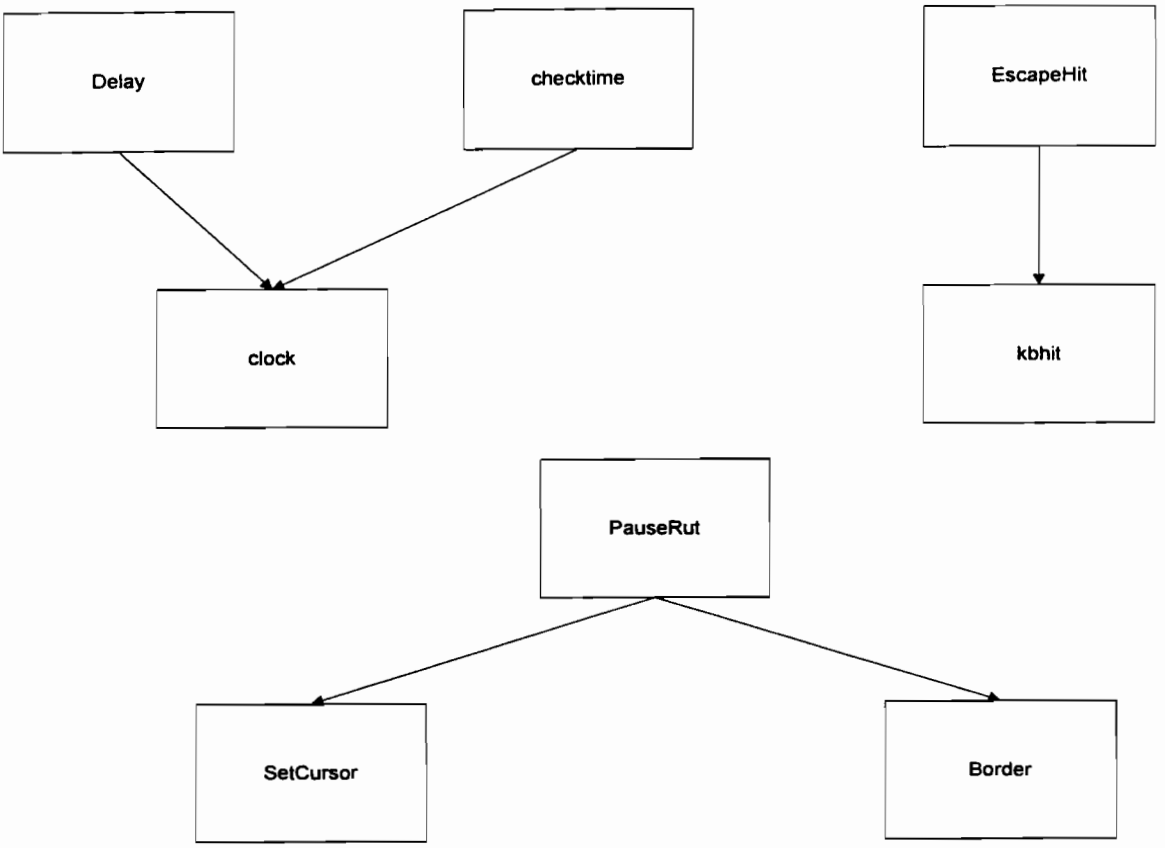


Level 5

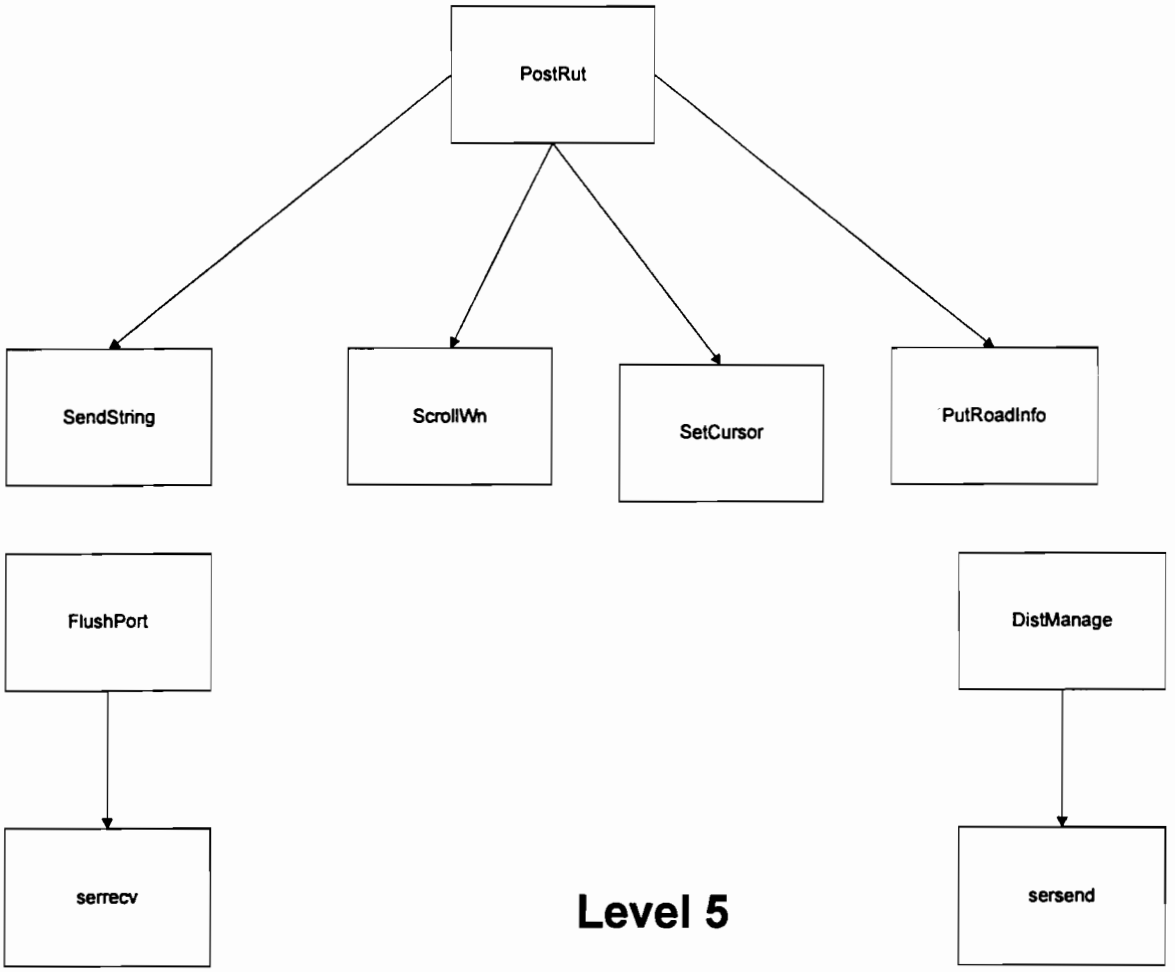

GetAcoustic
ClearRows
SetCursor
SelectAcoustic


SetAcUnit
FlushPort
SendCommand
GetString
sersend

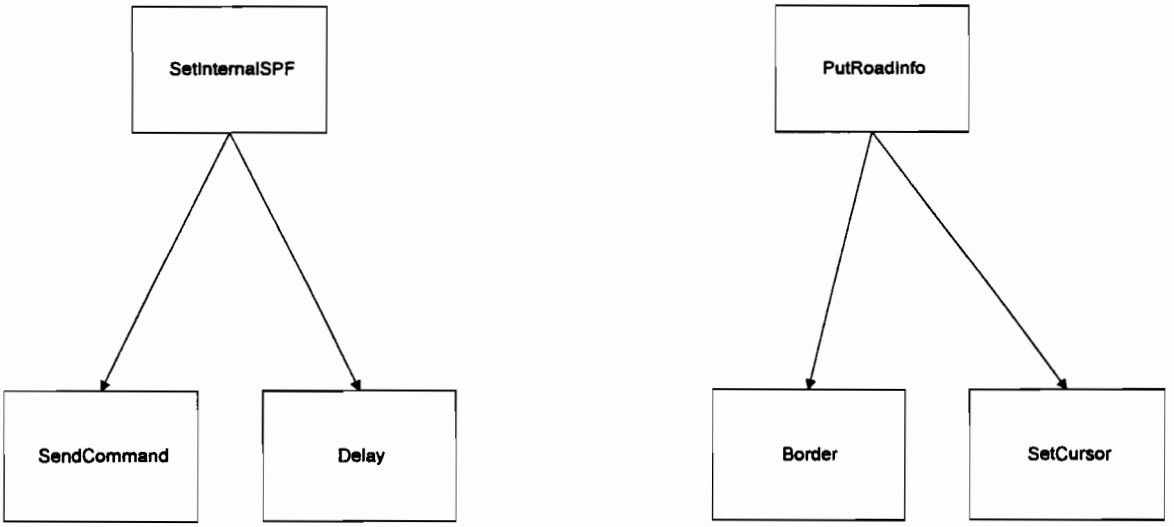
Level 5

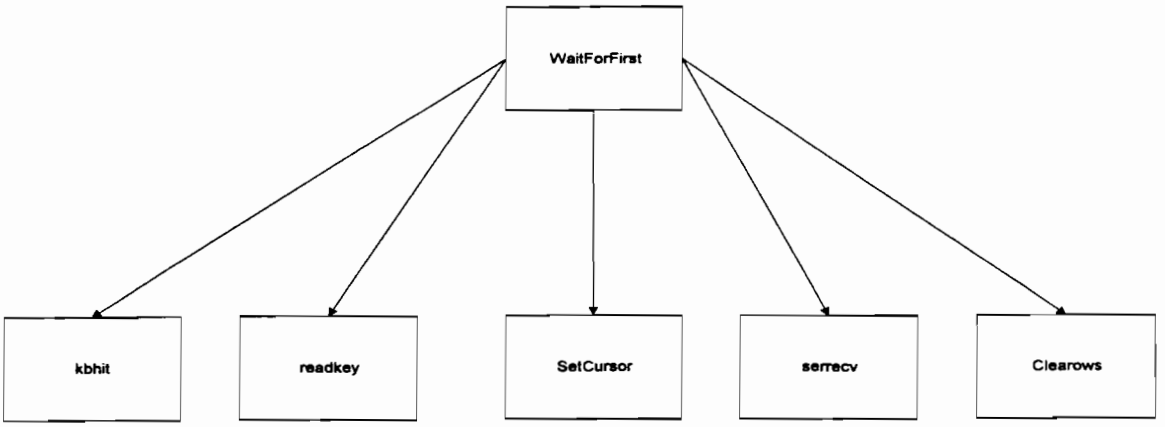

Delay
checktime
EscapeHit
clock
kbhit
PauseRut
SetCursor
Border

Level 5

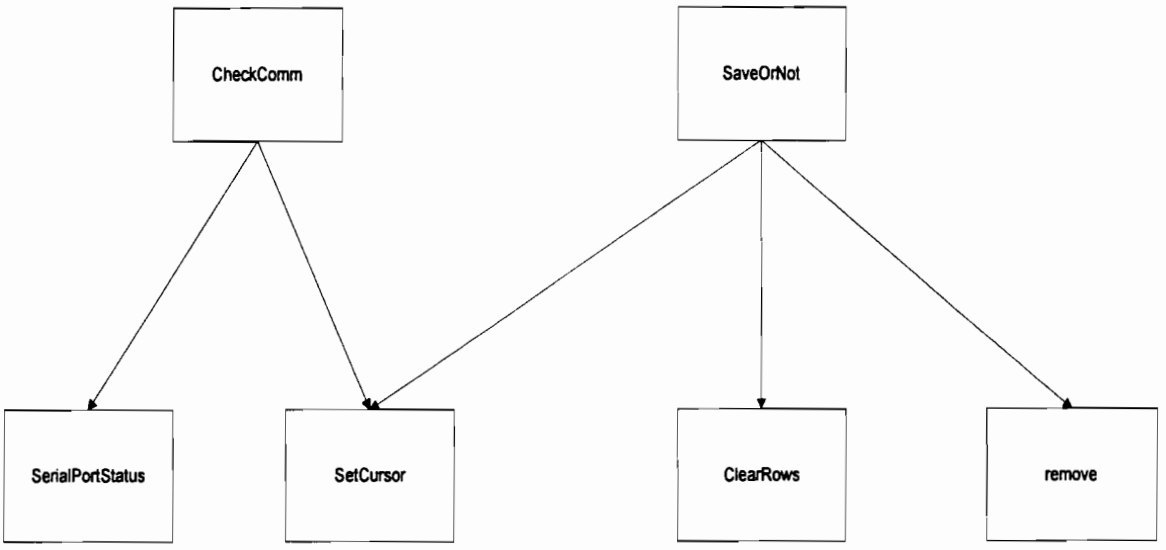

PostRut
SendString
ScrollWn
SetCursor
PutRoadInfo
FlushPort
DistManage
serrecv
sersend

Level 5

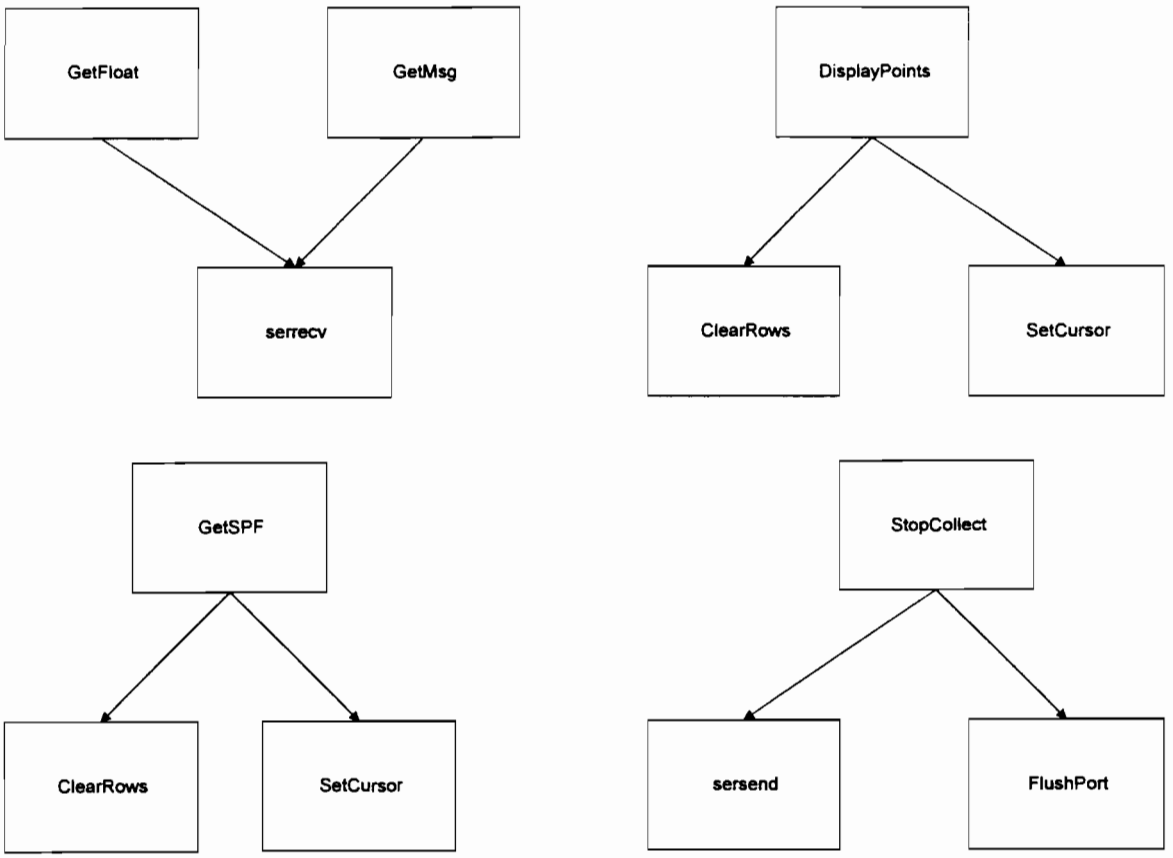

SetInternalSPF
PutRoadInfo
SendCommand
Delay
Border
SetCursor



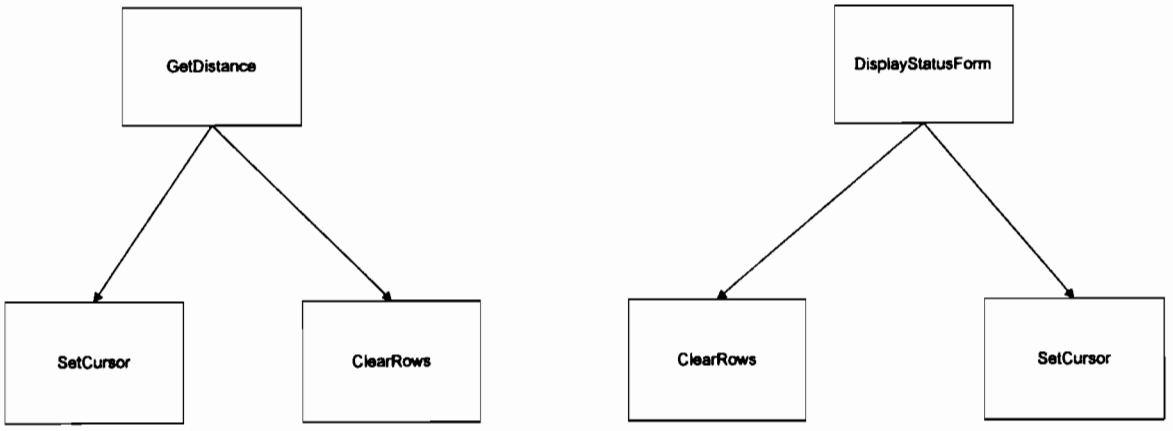
Level 5



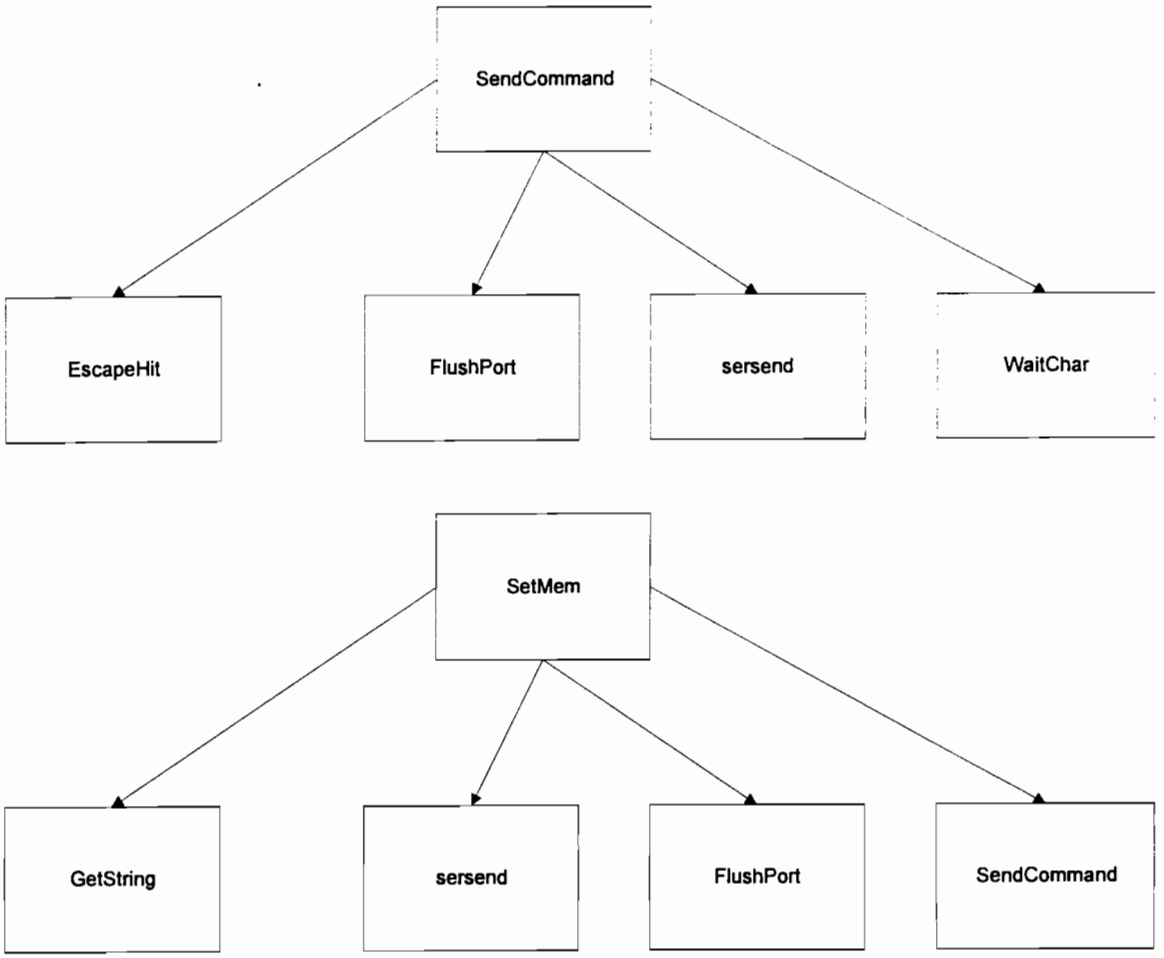
Level 5



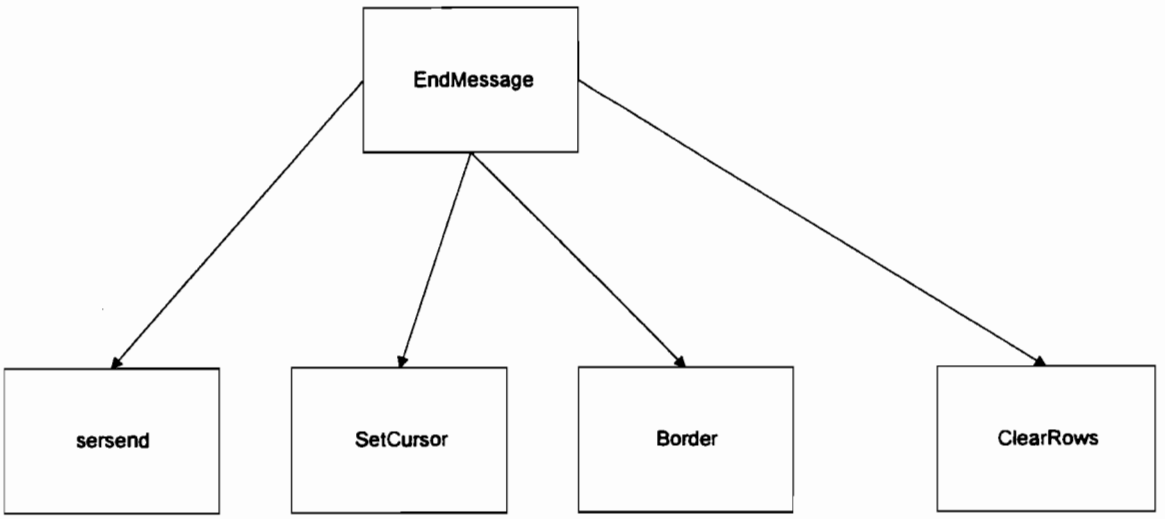
Level 5



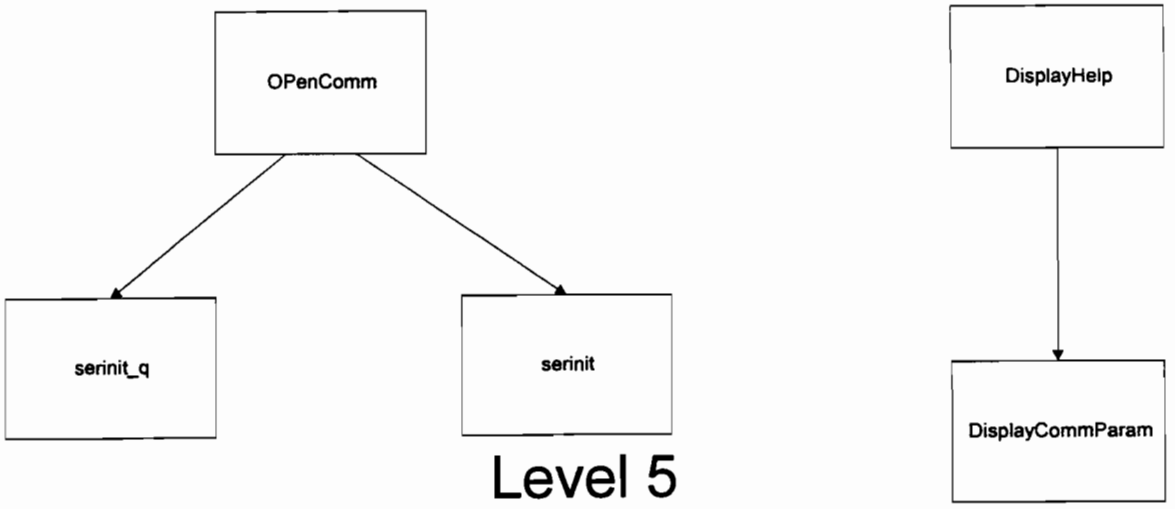
Level 5



Level 5



Level 5


OPenComm
serinit_q
serinit
DisplayHelp
DisplayCommParam
Level 5

Function Headers

Description of Talk Functions sorted alphabetically

Function name: *Border*

Input Parameters:

int lr	Upper left row Used
int lc	Upper left column Used
int rr	Lower right row Used
int rc	Lower right column Used

Return Value:

None

Global Variables Set and Used:

None.

Description:

This routine draws a border on the screen. It uses the horizontal, vertical, corner characters to draw the border.

Functions Called:

SetCursor
WriteCh.

Called by:

GetDistance
IRI_RUTMode
ProfileMode3
ProfileMode2
KeyPadMode
TestMode
PauseRut
SDProfile
EndMessage
PutRoadInfo

Example Calls:

Border(21, 2, 23, 14).

Function name: *CheckComm*

Input Parameters:

None.

Return Value:

None

Global variables set and used:

PortNo	Used
iOverrun	Set
	Used
iFrame	Set
	Used

Description:

This function calls the function SerialPortStatus. The function SerialPortStatus returns two bytes which describe the status of the port.
The Overrun error is checked with the help of this 16 bit information and the FrameError is checked with the help of this 16 byte information. There is a counter iFrame which is incremented if there has been a Frame Error. There is a counter iOverRun which is incremented if there is an Overrun error.

Functions Called:

SerialPortStatus, SetCursor.

Called by:

ProfileMode2, ProfileMode3, IRI_RUTMode and SDProfile1

Example Call:

CheckComm()

Function name: *CheckTime*

Input Parameters:

bReset	Used

Return Value:

0 | bdiff	bdiff is an unsigned integer.

Global variables set and used:

None

Description:

CheckTime(TRUE) indicates that the clock is reset and 0 is returned as the return value.
The variable CurrClock indicates the time when the most recent call to this routine is made.
The variable OldClock indicates the (last but one) most recent call to this routine is made.
CheckTime(FALSE) checks the difference between CurrClock and OldClock.
If the difference is greater that 1000 then this routine returns the difference (diff).
If the difference is lower that 1000, then a 0 is returned.

By default, the old clock is set equal to the current clock i.e. OldClock = CurrClock.
1000 indicates 1 second because the value returned by clock( ) is in milliseconds.
Observe that OldClock is defined as static.

Functions Called:

clock ( ) - C function

Called by:

ProfileMode2

Example Call

checkTime(True)
checkTime(False)

Function name: *CloseRut*

Input Parameters:

None

Return Value:

None

Global variables set and used:

fRut	Used
fSI	Used
fWSV	Used
iRutCnt	Used
iSICnt	Used
iAFirst	Used
iALast	Used
iBadAcou	Used
i	Set
	Used

Description:

This routine checks if the files for Rut, SI , WSV are open...that is 3 files. It writes Rut and all the bad acoustics to the file for Rut. It writes SI to the file for SI. It writes WSV to the file for WSV.
It closes all the files.
Functions Called: fprintf
fclose

Called:
None

Called by:

ProfileMode3

Example Call

CloseRut( )

Function Name: *CommParam*

Input Parameters:

None.

Return Value:

None.

Global Variables Set and Used:

BaudRate	Set
SupportedBaudRate	Used
Parity	Set
DataBits	Set
StopBit	Set
PortNo	Set

Description:

This function allows the user to select the communication parameters. It allows the user to select the communication parameter Baud Rate from any of the following Baud Rate, like { 9600, 4800, 2400, 1200, 600, 300, 150, 19200 }. The user is allowed to select the parity, the parity can be either No Parity, Odd Parity or Even Parity. The user is also allowed to select the number of stop bits either 1 or 2. The user is also allowed to select the number of databits 7/8.
The user can also select the ports, either COM1 or COM2.

Functions Called:

readkey

Called by:

ProcessFunctions

Example Calls:

CommParam()

Function name: *Delay*

Input Parameters :

ms	Time in milliseconds Used

Return Value:

one

Global variables set and used:

None

Description:

It just waits for time specified in (ms) , time in milliseconds and returns.

Functions Called:

clock ( ) - C function

Called by:

SetInternalSPF and TestMode

Example Call:

Delay(10L)

Function name: *DispCorr*

Input Parameters:

None

Return Value:

None

Global variables set and used:

SMountCorr	used
RuttCorr.Mounting	used
RuttCorrLoading	used
SLoadCorr	used
STotalCorr	used

Description:

It displays the variables SMountCorr, RuttCorr.Mounting, RuttCorrLoading, SLoadCorr & STotalCorr.
where
char sMountCorr[]="\nCurrent mounting corrections (inch):\n";
char sLoadCorr[] ="\nCurrent field loaded/loaded corrections (inch):\n";
char sTotalCorr[]="\nTotal corrections (counts):\n";
It displays the values of RuttCorr.Loading and RuttCottMounting for all the 5 acoustic devices. Then the program displays the RuttCorrMounting + RuttCorrLoading for all the 5 acoustic devices which is the total
Rutt Correlation for the 5 devices. The loading correlation is added to the mounting correlation.

Refer to **datadic.doc** for definitions of RutCorr.Mounting, RutCorrLoading, StotalCorr

Functions Called:

printf DOS

Called by:

LoadedCorr( )

Example Call:

DispCorr( );

Function name: *DisplayCommParam*

Input Parameters:

None

Return Value:

None

Global Variables Set and Used:.

PortNo	Used
BaudRate	Used
Parity	Used
Databits	Used
StopBits	Used

Description:

This function displays info. about the communication parameters. It displays the parity; whether the parity is ODDPARITY, EVENPARITY or NOPARITY along with the number of databits and the number of stopbits.

Functions Called:

None

Called by:

ProcessFunctions, DisplayHelp

Example Calls:

DisplayCommParam()

Function name: *DisplayHelp*

Input Parameters:

None

Return Value:

None

Global variables set and used:

None

Description:

It gives us the option of which keys need to be pressed to change communication parameters, Toggle between hex and ascii mode, select data collection mode, Metric/English, Test Acoustic devices, Siometer Panel, Profile Mode and to exit. The existing communication parameters are displayed by the call to the function DisplayCommParam. This helps the use to see the existing parameters and decide what he/she needs to do next and change which parameters.

Functions Called:

printf
DisplayCommParam

Called by:

main

Example Call:

DisplayHelp

Function name: *DisplayPoints*

Input Parameters:

int iRow	The rows are used as a parameter for clearing the screen and setting the cursor
	Used
long nPts	Used
double dist	Distance
	Set
	Used
double sample	Samples
	Used

Return Value:

None

Global variables set and used:

bMetric Used

Description:

This function clears the rows and sets the cursor. If the variable bMetric is set the distance is printed in kilometers otherwise the distance is printed in miles and feet. Also the number of points and samples are also printed.

Functions Called:

ClearRows
SetCursor

Called by:

ProfileMode2

Example Call:

DisplayPoints(11, nPts, dist, sample);

Function name: *DisplayStatusForm*

Input Parameters:

int iRow Row
Used

Return Value:

None

Global Variables Set and Used:

bMetric Used. It is a Boolean variable. If the basic metric is KM then it is TRUE else if it is FEET then it is FALSE. [set/used and What is the significance of the variable]

Description:

It clears the screen and prints the distance traveled in KM if the bMetric is TRUE else it prints in FEET or in MILES if bMetric is FALSE.
This function prints a form for the Time Elapsed and Distance traveled. It prints the literal strings "Busy = /1000". Observe the blank space between "Busy" and the "/1000". This space is conserved for the cursor to be set later on and some data to be printed in the position between "Busy " and "/1000". Also the cursor is set and the strings " Time Elapsed= " and " Distance Traveled = ", " km", " feet (miles)" are printed. The variable bMetric is checked. If this variable is TRUE then the "km" literal string is printed otherwise the " feet (miles)" literal string is printed.

Functions Called:

ClearRows
SetCursor.

Called by:

IRI_RUTMode
ProfileMode2
ProfileMode3
SDProfile.

Example Calls:

DisplayStatusForm(int iRow).

Function name: *DistanceMode*

Input Parameters:

bDisplay	It is of the Boolean type. So it can take values of TRUE | FALSE Used.

Return Value:

int IsDistance	This tell us whether it is DistanceMode or TimeMode If IsDistance is TRUE then it is DistanceMode else it is TimeMode.

IsDistance = TRUE | FALSE

Global Variables Set and Used:

None

Description:

This routine sends the command "D 42EE" using the SendCommand function. The routine then gets a string from the port which is either TRUE or FALSE. The routine determines whether IsDistance is TRUE | FALSE. If IsDistance is TRUE then it prints "DistanceMode " otherwise it prints "TimeMode".

Functions Called:

SendCommand, GetString, sscanf, SetCursor, printf

Called by:

ProfileMode2
DistanceMode

Example Calls:

DistanceMode(TRUE), DistanceMode(FALSE).

Function name: *DistManage*

Input Parameters:

int iFunc where iFunc ⊂ { DM_RUT, DM_SI, DM_INIT }
Used

Return Value:

DISTTYPE DistR

Global Variables Set and Used:

nRutsPerDMI	Used
bExtraComm	Used
RutDist	Used
SIDist	Used

Description:

This routine first finds out the value of iFunc. The value of iFunc is determined. iFunc is either DM_RUT, DM_SI, DM_INIT. Depending upon the value of iFunc a certain steps of functions is performed.
Why do you need this function ?
The mode3 obtains the value of SI every 0.1/0.2 mile, that is every 1056 feet or 528 feet depending upon the setting. The mode3 also obtains the value of the acoustic data every 4 feet. The program running on the PC gets the acoustic data as well as the SI from the siometer.
In this program we need to send Distance Signals to the mission manager if COM3 is connected.
Since the acoustic data comes in every 4 feet we can send a distance signal every 4 feet. But the acoustic data is not a true representative of how much distance is covered. The SI is a true representative of how much distance is covered. But the acoustic data comes in every 4 feet whereas the SI comes in every 1056 feet. Whenever acoustic data comes in a distance signal is sent to the mission manager.But you know that this acoustic data arrival does not really allow you to send a distance signal after exactly 4 feet. Therefore you could have sent either more or less distance signals. Therefore you need to do some corrections. The SI comes in every 1056 feet. Therefore when the SI signal comes in you now whether you need to send some more distance signals or lesser.

Whenever this routine is called using DM_INIT it means that this routine is being called for the first time by mode3
Whenever this routine is called using DM_SI i.e. DistManage(DM_SI) it means that a value of SI has come in and we need to adjust the sending of distance signals.
Whenever this routine is called using DM_RUT i.e. DistManage(DM_RUT) it means that a value of RUT , acoustic data has come in and we need to adjust the sending of the distance signals

Lets see how we do it in this routine.

- This routine is called with DM_INIT whenever it is called for the first time to initialize the values of the variables used in this routine.
  The following initializations are done in this routine if it is called with DM_INIT.
  DistR = 0;
  DistR is the cumulative of the total distance covered so when this routine is initially called, DistR is set to zero.

iDCnt = 0;
iDCnt is the number of times the routine has got acoustic data so it is initially set to zero.
DistBalance = SIDist
SIDist is either 1056 or 528. this value is in feet.

- This routine is called with DM_RUT whenever mode3 gets a RUT i.e. (acoustic data).

Whenever a RUT comes in that is acoustic data comes in the value of iDCnt is incremented.
Observe that the type of iDCnt is static. Once it is initialized, it remains the same no matter what the value of iDCnt is.
When the value of iDCnt reaches nRutsPerDMI (3) , then if DistBalance = 0 and bExtraComm is 1, a command "D" is sent to the serial port COM3. The RUT comes in every 4 feet. So 4 * 3 = 12 feet. Every 12 feet a "D" command is sent to COM3. Also the adjustment of the distance signals is done. DistR is the cumulative of all the distance covered. (DistBalance is initially set to SIDist by the initialization routine.)
DistBalance is decremented by the amount of total distance covered since the last SI signal.

- This routine is called with DM_SI whenever mode3 gets an SI i.e. (This routine is called with DM_SI whenever mode3 gets an SI i.e. (Serviceability index).

Whenever an SI value comes in, that is every 1056, 528 feet, if lesser number of distance signals have been sent the correction is made and the exact amount of distance signals which were supposed to be sent depending the distance covered are sent. This correction is made here because SI gives us an indication of the exact distance covered.

Observe the while loop below. Until DistBalance does not become zero, a correction is made, that is the distance signal is sent. After the DistBalance becomes zero, the exact number of distance signals have been sent.
The DistBalance is initialized to SIDist which has to be decremented every time a RUT ...DM_RUT comes in.

```
// control for tow few samples
while (DistBalance > 0) {
  // Send out extra D's to COM3
  if (bExtraComm) sersend(COM3, 'D');
  DistR += RutDist*nRutsPerDMI;
  DistBalance -= RutDist*nRutsPerDMI;
}
DistBalance += SIDist;
```

Functions Called:

Sersend

Called by:

ProfileMode3

Example Calls:

DistManage(DM_INIT)

Function name: *EndMessage*

Input Parameters:

None.

Return Value:

None.

Global Variables Set and Used:

PortNo	Used
bAbort	Used

Description:

It prints the end message. If sending has been aborted then the it prints as 'ABORTED' else if sending has been completed then it prints 'COMPLETED' and asks you to press any key to proceed. It finally clears the screen.

Functions Called:

sersend
SetCursor
ClearRows
Border
Readkey

Called by:

IRI_RUTMode
ProfileMode2
ProfileMode3
SDProfile.

Example Calls:

EndMessage().

Function name: *EscapeHit*

Input Parameters:

None

Return Value:

BOOLEAN TRUE | FALSE (TRUE is returned when the ESCAPE key is hit. FALSE is returned when the ESCAPE key is not hit)

Global variables set and used:

None

Description:

Checks whether an Escape key has been hit on the Keyboard. If Escape key has been hit it will return TRUE else it return FALSE to the calling function.

Functions Called:

kbhit

Called by:

Upload
SendCommand
GetString
ProcessFunction

Function name: *FlushPort*

Input Parameters:

int PortNo PortNumber

Return Value:

None

Global variables set and used:

None

Description:

The function call to serrecv is made again and again until a word FFFF (hex) is received.
The event of receiving a word FFFF indicates that the port has been flushed...cleared.

Functions Called:

serrecv

Called by:

SetSdMode, SendCommand, GetString, Upload, ProcessFunctions, KeyPadMode, TestMode, SetAcUnit, StopCollect, ProfileMode3, SetMem

Example Calls:

FlushPort(COM1)

Function name: *GetAcoustic*

Input Parameters:

int Row	Used
int *acunit	Used

Return Value:

None

Global Variables Set and Used:

szDefAc	Set
	Used
szActiveAcoustic	Used

Description: The function 'GetAcoustic' asks the user enter the selected acoustic devices. It first clears the rows and sets the cursor in a particular position on the screen. It prints a statement "Enter selected acoustic device(s):" Here the user is supposed to enter the numbers of the acoustic devices that should collect the data. The acoustic devices are numbered 1 through 5 with an optional laser that can be set and given the number 6 to select. It prints on the screen the numbers of the acoustic devices that have been selected for the collection of the data.

Functions Called:

ClearRows
SetCursor
SelectAcoustic

Called by:

ProfileMode3
ProfileMode2
IRI_RUTMode
SDProfile1

Example Calls:

GetAcoustic(10, &acunit)

Function name: *GetCursor*

Input Parameters:

int *r	The rows position
	Set
int *c	The column position
	Set

Return Value:

None

Global variables set and used:

None

Description:

Reads the current cursor position for a specific display page and the current cursor size. It uses INTERRUPT 0x10h - Function 03h (GET CURSOR POSITION AND SIZE). For this AH is 03, BH is page number, DH is the row(00h is top), DL is Column (00h is left).

Functions Called:

int86 INTERRUPT 0x10h - Function 03h

Called by:

[Need to find which functions are calling this function]
At present we did not find any function calling this function.

Example Calls:

Obviously None

Function name: *GetDistance*

Input Parameters:

int Rows	Used
char *unit	Used
	Set
double *dist	Set
	Used

Return Value:

None

Global Variables Set and Used:

bMetric	Used	
str	Set	This is a buffer.

Description: This function basically gets distance from the user. This function asks the user to enter the distance. As long as the distance is 0 or less it sets the cursor and clears the rows. Then it prints a statement 'Enter distance'. Here the user has to enter the distance. This function has to get the distance from the user. If the distance is not entered it prompts again and again the user to enter the distance.

Functions Called:

SetCursor
ClearRows

Called by:

Mode6
IRI_RUTMode
ProfileMode3
ProfileMode2

Example Calls:

GetDistance(12, &unit, &distance)

Function name: *GetFileName*

Input Parameters:

iRow	Int type
	Set
	Used

Return Value:

A pointer to the file

Global Variables Set and Used:

bSysGenFileName	Used
str	Set
	Used
Datfile	Set
	Used
bComment	Used
Header	Used
	Set
SI_PACK	Used
DataCollectMode	Used
SI_AVGSPD	Used

Description: It asks the user to enter the name of the output file. If the user does not enter a name of the file then the program itself generates a name of a file using the system time. It uses the localtime function that is supported by the 'C' library to get the local time[actually localtime function converts calendar time into local time] to generate the name of the file. The name of the output file name is stored in the variable 'DatFile'. To verify whether the file by the name already exits we try to open the file in the read mode. If it already exits then the 'fopen' function returns a non-null value. If it is so then we close the file and tells the user " File already exists. Replace?[N]/Y:" to give another name for the file. Then it prints the name of the output file on the screen. It prints the header and allows the user to enter 1 or 2 lines of comment.

Functions Called:

ClearRows
SetCursor

Called by:

IRI_RUTMode
ProfileMode2
ProfileMode3
SDProfile1
Mode6

ExampleCalls:

GetFileName(6)

Function name: *GetFloat*

Input Parameters:

None

Return Value:

float

Global Variables Set and Used:

PortNo Used

Description: This function uses serrecv to receive data from the serial port denoted by PortNo. The data received for first time is stored in a variable. Then the port is read for 4 more times. Each time the serial port is read zeros are introduced into the lower order bits of data in multiples of 7 and same number of highest order bits are lost by shifting to left. For the data received fist 7 higher order bits are shifted out and 7 zeros are shifted into lower bits. Then this data is bitwise 'OR'ed with data that is stored in the variable and the result is stored in the variable. Next time the data received will be 14 higher order bits are shifted out introducing 14 zeros into the lower order bits. Again this is bitwise 'OR'ed with the data stored in the variable ad the result is again stored in the variable. This process is followed for the 3rd and 4th time with 21 and 28 zeros respectively introduced into the data that is received on 3rd and 4th time. Then the value in the variable is returned.

Functions Called:

serrecv

Called by:

RutMode

Example Calls:

GetFloat

Function name: *Get_Header(char *Filename)*

Input Parameters:

Filename That is a pointer to a char array containing the filename . The function prototype is defines as shown. [char *GetHeader(char *FileName)]

Used

Return Value:

char * Pointer to a character

Global variables set and used:

None

Description:

It gets the 80 character header from the filename provided as input and adds a \0 character to the end of the header string. It returns the 80 character header.

Functions Called:

fopen
malloc
free
fclose
fgets

Called by:

InitSi

Example Call:

GetHeader(value)

Function name: *GetInternalSPF*

Input Parameters:

None

Return Value:

SPF An integer

Global Variables Set and Used:

None

Description: This function sends "D4E3E" to the siometer. Then it reads the siometer by using the GetString function. As already explained in SetMem the format of the siometer is "Prompt <address> <word value in Hex>" The word value in Hex starts in 9 th byte. This value (that is data at the 9th byte and after) is copied into the buffer that is returned when this function is called.

Functions Called:

SendCommand
GetString

Called by:

ProfileMode2

Example Calls:

GetInternalSPF()

Function name: *GetIRQ*

Input Parameters:

PortNo	Port number Used

Return Value:

Integer	Interrupt Number

Global variables set and used:

iCOM3IRQ	Used.
PortNo	Used.

Description:

Depending on the port number it returns an integer. If port number = COM1 it returns 4, if it is COM2 it returns 3, if it is COM3 it returns iCOM3IRQ else if it is COM4 it returns 2.
After the program has executed the ports accessed by the interrupt's are fixed. The number which is returned is an integer which is the interrupt number of the port . The port number is given as an input parameter.

Functions Called:

None

Called by:

serinit_q
serclose-q.

Example Calls:

GetIRQ(COM2)
GetIRQ(COM3)

Function name: *GetMem*

Input Parameters:

long lAddr	It is an address Used

Return Value:

int iValue	It contains an address .

Global Variables Set and Used:

None.

Description:

It accepts even addresses only. This function copies the given input parameter "lAddr" to the buffer "s" using the sprintf command. Therefore, if the example call to GetMem is GetMem(0x10529eL)
Then the sprintf statement will cause the buffer " s " to have the contents "10529E".
The function SendCommand sends the string "10529E" to the global port PortNo.
The function GetString then receives a string on the same port.
The address of the 10th character is copied into iValue.
The iValue is returned.

Functions Called:

SendCommand
GetString.

Called by:

ProfileMode3.

Example Calls:

GetMem(long lAddr)

Function name: *GetMsg*

Input Parameters:

char *s Set

Return Value:

None

Global Variables Set and Used:

PortNo Used

Description: The function 'GetMsg' receives the message on the serial port denoted by port PortNo.

Functions Called:

serrecv

Called by:

IRI_RUTMode

Example Calls:

GetMsg(str)

Function name: *GetRut*

Input Parameters:

r1	Pointer to a floating type Set
r2	Pointer to a floating type Set
acou	Array of short integer Used

Return Value:

None

Global Variables Set and Used:

iRutMethod	Used
SOUTH_DAKOTA	Used
CENTER	Used
STRIGLINE_R	Used
RIGHTRUT	Used
STRINGLINE_L	Used
w1	Used
w2	Used
w3	Used
w4	Used
COUNTS_PER_INCH	Used

Description: Different methods of rut measurement use different data collected to measure the rut. Depending on the type of the Rutmethod employed, certain calculations are performed. The different types of rut measuring methods are SOUTH_DAKOTA method, STRINGLINE_R (String line Right) method, STRINGLINE_L(String line Left) method, RIGHTRUT(Absolute right) method and stringline method.

Functions Called:

COUNT_TO_INCH

Called by:

ProfileMode3

Example Calls:

GetRut(&r1, &r2, iAcou)

Function name: *GetSCKPMI*

Input Parameters:

None

Return Value:

Integer

Global Variables Set and Used:

None

Description: This Function sends a string "D 4E3C" to the siometer. It reads a string from the siometer. In the sscanf function it has been offset by 9 as the first eight bytes is made up of prompt and the address. From the 9th byte we have the word value in Hex. This is returned.

Functions Called:

SendCommand
GetString

Called by:

Need to find out which functions are calling this function.

Example Calls:

None

Function name: *GetShort*

Input Parameters:

None

Return Value:

integer

Global Variables Set and Used:

PortNo Used

Description:

This function 'GetShort' reads the serial port twice. The data received from the serial second time is shifted 7 times to left introducing 7 zeros in to the lower order bits. In this process 7 highest order bits of data are removed and each bit in the data shifts 7 positions to its left. Then bitwise 'OR' is applied on the data that was received first and the one that has been shifted (i.e. one that is received second) The truth table of bitwise 'OR' operation is as follows

A	B	T
0	0	0
1	0	1
0	1	1
1	1	1

From this table we see that if one of it or both of them is 1 then the truth value is 1. Another important thing to notice is that when the value of B is 0 then the truth value is same as the value of A. So 'OR'ing data say A with another data B that has all 0's in the bits does not change the value of A.
So here by introducing 7 zeros in to the word that is received second we are not changing the 7 lower order bits of the data that is received first. This value is returned to the calling function.

Functions Called:

serrecv

Called by:

RutMode

Example Calls:

GetShort()

Function name: *GetSPF*

Input Parameters:

iRow	int type
	Used
Mode	int type
	Used
*f	pointer to a file (FILE type)
	Used

Return Value:

double

Global Variables Set and Used:

ACOUSTIC	Used
ACCELERATION	Used
SI_AVGSPD	Used
iSIModeFPS	Used
SI_PACK	Used
str	Used
	Set
bCountAcoustic	Used

Description:

This function clears the rows and sets the cursor every time. If the mode is SI_AVGSPD then it prints the distance between the samples else it asks the user to enter the it. To exit this function the user has to enter a value less than or equal to zero.

Functions Called:

Called by:

Example Calls:

Function name: *GetString*

Input:

char *s	The address of the buffer where the string is to be received
	Set
int iLen	The length of the string to be received
	Used

Return Value:

None

Global Variables Set and Used:

bAbort	Set
PortNo	Used

Description:

This routine gets a string of length iLen in the specified buffer s. The address of the buffer is provided to the routine as an input parameter *s. The parameter iLen is the number of characters to be read into the buffer s. The routine WaitChar waits for a character and times out if a character does not come in . This routine used the routine WaitChar for waiting on every character of the string coming in.
After the whole string of the specified length iLen has come in, then the character \0 is attached to the end of the sting to make it a proper string....end of string.

Functions Called:

WaitChar, EscapeHit and FlushPort.

Called by:

SetMem, GetMem, SetSdMode, SetAcUnit and DistanceMode.

Example Calls:

GetString(s , sizeof(s)-1)

Function name: *InitRut*

Input Parameters:

None

Return Value:

None

Global Variables Set and Used:

iBadAcou	Set	iRutMethod	Used
SOUTH_DAKOTA	Used	iAFirst	Set

L1	Used	iALast	Set
R1	Used	CENTER	Used
R2	Used	L2	Used
STRINGLINE_R	Used	STRINGLINE_L	Used
STRINGLINE_RL	Used	RIGHTRUT	Used
NEWAVERAGE	Used	cROADINFO	Set
iDistance	Set	iRutCnt	Set
iRutLine	Set	iSICnt	Set
iSILine	Set	Range	Used
iSum	Set	nSamples	Used
bExtraComm	Used	fRut	Set , Used
fSI	Set,Used	fWSV	Set, Used
iInterval	Set,Used	iSIModeFPS	Used
RangeLimits	Set	iRutSecLen	Used

Description: This function initializes the variables. Depending on various rut methods like SOUTH_DAKOTA, STRINGLINE etc., it sets the variables like iAFirst, iALast differently. It basically sets the variables that are used in for measuring the Rut.

Functions Called:

None

Called by:

PriofileMode3

Example Calls:

InitRut()

Function name: *InitSi*

Input Parameters:

fn	Pointer to character Used

Return Value: None

Global Variables Set and Used:

iCOM3IRQ	Set Used	iRutSecLen	Set
iRutMethod	Set Used	bExtractComm	Set
Header	Set Used	bSysGenFileName	Set
iSIModeFPS	Set	DataCollectMode	Set
szDefAc	Set		

	Used		
Param	Set	str	Set
	Used		
value	Set	bF5	Set
	Used		
bF2	Set	BaudRate	Set
iMode6FPS	Set	RutCorr.Mounting	Set
fn	Used		
	[This is a file]		

Description: This function opens the file whose name is stored in variable called 'fn' in read mode. Then it reads each token and compares the token for various predefined tokens. Each time two tokens are read, the first token is read into variable called 'param' and the next one into variable called 'value'. The tokens stored in the variable 'param' are compared to predefined tokens and when the token read is same as any predefined token then a particular variable is set. For instance if the token read from the file is "FPS" then the variable iSIModeFPS is set to the integer value of the value stored in the variable 'value'. This process of reading tokens from the file continues until the whole file has been read. The initialization file is read. This filename is supplied as a command parameter to talk70.exe. **Example:** Declarations in the initialization file are:

CollectMode = 3
header = select.hdr
acoustic=12345
fps=4
gen_filename = Y
comment = N
F5 = N
F2 = N
baud = 9600
port = 2

Functions Called:

GetHeader

Called by:

Main

Example Calls:

InitSI(argv[1])
InitSi(" ")

Function name: *KeypadMode*

Input Parameters:

None

Return Value:

None

Global Variables Set and Used:

PortNo	Used
_GCLEARSCREEN	Used
_TEXTBW40	Used
GCLAKEY	Used
CALIBKEY	Used
KEY9	Used
STOPKEY	Used
RUNKEY	Used
TESTKEY	Used

Description: This function sends "J3FF0" on a given port. Then it flushes the port, and calls _clearscreen and _setvideomode. Then it sets the cursor and prints '1 G Cal', '2 Caliberate', '3 Speed', ' 4 Stop', one below the other. Then next to these '5 Run', '6 Test' and 'ESC Exit' are printed. Then the keyboard is read constantly until an escape character(ASCII value 27) has been hit. For each key that has been hit on the keyboard, its ASCII value is stored in the variable key.ascii where key is of the scancode type. Depending on the key that has been hit on the keyboard the variable 'iKey' is assigned a particular value. If the value of the character that has been hit on the keyboard is in between '6' and '1' then it means that we wish to perform one of the above procedures namely Run, Test, Speed etc. This is sent to the siometer. Then the port that has been used for sending is read to receive a character from the siometer and it is stored in the variable called 'wTemp'. Then it sends escape character whose ASCII value is '27' and ASCII value '13' to the siometer. Then the function _setvideomode is called in the _DEFAULTMODE.

Functions Called:

SendCommand
FlushPort
_clearscreen
_setvideomode
SetCursror
Border
readkey
sersend
_setvideomode

Called by:

ProcessFunction

Example Calls:

KeypadMode

Function name: *LoadedCorr*

Input Parameters:

None

Return Value:

Int

Global Variables Set and Used:

RutCorr.Mounting	Used
PortNo	Used

Description: This function reads the rut correction by calling the procedure ReadRutCorr. Then it displays it by calling the procedure DispCorr. It prompts on the standard output (normally the console) "Press [F7] to do the unloaded condition." If the key F7 is pressed then it goes ahead and does the for unloaded condition. At this point of time all the occupants including the driver should get out of the vehicle. This is done because the rut bar's inclination should be known at the exact time of the acoustic data sampling. Then it would be possible to correct acoustic readings so that they match the accurate readings. To abort this press escape, to repeat press function key 6 (F6) or press function key 7 (F7) to accept the collected data. Then again the statement "press [F7] to do loaded condition" is printed. If we press the key F7 then it goes ahead and collects the data for the loaded condition. Then it prints on the standard output "ESC- Abort [F6]- Repeat [F7] - Accept. This statement tells the user to press escape key to abort F6 key to repeat the collection of the data and F7 to accept the collected data. If F7 is pressed on the keyboard then the collected data is saved in file by calling the procedure SaveRutCorr.

Functions Called:

ReadRutCorr
DispCorr
readkey
TestMode
SaveRutCorr
SetCursor
sersend

Called by:

ProcessFunction

Example Calls:

LoadedCorr()

Function name: *Main*

Input Parameters:

argc	int type It is basically the number of arguments in the command line.
argv	pointer to an array of pointer

Return Value:

int type

Global Variables Set and Used:

ASCIIMODE	Used
DisplayMode	Set
PortNo	Set
BaudRate	Set
Parity	Set

Databits	Set
StopBits	Set
szMode	Set
Header	Used
COM1	Used

Description: This function sets the parameters like the baud rate, port number, number of data bits, number of stop bits parity.

Functions Called:

InitSI
ReadRutCorr
DisplayHelp
MonitorMode
_clearscreen

Called by:

None

Example Calls:

None

Function name: *MasterOutput*

Input Parameters:

szFormat	Char
	Used
wVal	unsigned
	Used
bPreSection	Bool
	Used

Return Value:

None

Global Variables Set and Used:

nInLine	Set
fData	Set
mph	Used
fps	Used
TWOCH	Used
cRoadInfo	Set
	Used
DataCollectMode	Used
CONSTR	Used
SI_AVGSPD	Used

Description: This function basically uses the file opened by the file pointer fData. It prints the speed. (If mph is not equal to -1 then the speed is in miles per hour. Else if fps is not equal to -1 then speed is in feet per second) Then values stored in the variables 'szFormat' and 'wVal' are stored in the file. The value stored in the variable 'cRoadInfo' is also copied into the file. At the end a '\n' character is copied into the file. This file has a particular format. It has 16 values per line.

Functions Called:

None

Called by:

ProfileMode2

Example Calls:

MasterOutput("%04X", aVal, bPresection)
MasterOutput("Y%03X", aVal, bPresection)

Function name: *MonitorMode*

Input Parameters: None

Return Value: None

Global Variables Set and Used:

PortNo	Set
	Used
Parity	Used
DataBits	Used
StopBits	Used
COM1	Used
COM2	Used
BaudRate	Used
DisplayMode	Used
PF4	Used

Description: This function sets the port No. variable to COM1 serial port. Then it opens the ports 'COM2' and 'COM3' for communication at the baud rate of 9600. It opens the 'COM1' port at the selected baud rate, parity, databits and stopbits. Then it prints the statement "Connect the Siometer to COM1". This function makes use of the structure Scancode type. It is defined as follows.

```
typedef struct {
 unsigned char ascii;
  unsigned char scan;
} SCANCODE;
```

See Appendix A for more info on Scancodes
The variables ascii is used to store ascii value of the character and the variable 'scan' is used to store number of the key that generated the ascii character. When a key is pressed on the keyboard an electric impulse is generated

indicating the position of the key pressed. This impulse is handled by the keyboard processor which converts the impulse into number called scan code. This scan code stored in the variable scan that is defined in the structure above. As long as scan code is not PF4 read the key board by calling the procedure "readkey()". If the ASCII value is not equal to zero then send it serially to the siometer on the COM1. If the ASCII value is 'zero' then procedure "ProcessFunction()" is called and the window is scrolled up. Then a word is read from the serial port COM1 using the procedure "serrecv". If the word is '0xffff' then nothing is done. If it is '10' and the DisplayMode is ASCIIMODE then window is scrolled up by 1 line. If the received word is '13' and the DISPLAYMODE is ASCIIMODE then cursor is placed on line 23 and 0 column. If the received word is none of the above then the received word is printed on the screen. The above process will continue until key.scan is PF4.

Functions Called:

OpenComm
readkey
sersend
kbhit
ProcessFunction
ScrolWin
Serrecv
SetCursor
serclose_q
serclose.

Called by: Main

Example Calls: Monitor Mode().

Function name: *OpenComm*

Input Parameters:

PortNo	int type
	Used
BaudRate	int type
	Used
Parity	int type
	Used
DataBits	int type
	Used
StopBits	int type
	Used

Return Value:

Integer

Global Variables Set and Used:

uComAddr	Used

Description: This function sets the various elements of the structure defined below:

```
unoin{
    unsigned char c;
    struct {
            unsigned char databits :2;
            unsigned char stopbits : 1;
            unsigned char parity   :2;
            unsigned char baudrate :3;
          }b;
     } SERCONFIG
```

If the value in variable 'Databits' is 7 then databits variable in the above structure is set to 2. If the value stored is 8 then databits variable is set to 3. If it is none then variable 'Error' is set to -1. Similarly if the number of stopbits stored in the variable 'Stopbits' is 1 then the variable stopbits in structure above is set to '0' else if it is 2 then it is set to '1'. Similarly the baud rate and the parity are set.
This is a asynchronous serial communication. We need to send start and stop bits for each character that we send. The number of stop bits are normally 1, 1.5 or 2. The number of stop bits required normally depends on the extra time that the receiving device may require before it can start processing next character. The stop bits force a minimum gap between the successive frames. When two devices are set for communication they should agree on baud rate, #of stopbits, parity etc..

Functions Called:

Serinit

Called by:

MonitorMode

Example Calls:

OpenComm(COM3, 9600, Parity, DataBits, StopBits)
OpenComm(COM3, 9600, Parity, DataBits, StopBits)

Function name: *PauseRut*

Input Parameters:

None

Return Value:

None

Global Variables Set and Used:

bPause	Set
	Used

Description:

This function prints 'pause' and draws a border on the screen if bPause is true. If bPause is false then it just prints a blank.

Functions Called:

SetCursor
Border

Called by:

ProfileMode3

Example Calls:

PauseRut()

Function name: *PostRut*

Input Parameters:

None

Return Value:

None

Global Variables Set and Used:

iAFirst	Used	iALast	Used
iAcouHigh	Used	iAcouLow	Used
iBadAcou	Set	bBadISection	Set
nSamples	Set	RangeLimits	Used
iSum	Set	iDistance	Set, Used
iInterval	Used	iRutSecLen	Used
SRINGLINE_RL	Used	NEWAVERAGE	Used
bExtraComm	Used	COM2	Used
Range	Used	iStatus	Used
fBug	Used	r2	Used

Description:

This function determines whether the data that is collected is good or bad. If the raw data stored iniRaw is greater than iAcouHigh or lesser than iAcouLow then it is considered to be bad data. The value in iSum[1] are incremented if variable is lesser than RangeLimits[0]. If value stored in variable r1 is lesser than RangeLimits[2] then iSum[2] is incremented. If none of the above is satisfied then iSum[3] is incremented. Similarly if the value in variable 'r2' is lesser than RangeLimits[1] then iSum[1] is incremented. If it is lesser than RangeLimits[2] then iSum[2] is incremented. If noneof the above conditions are satisfied then iSum[3] are incremented. If the number of samples is zero then all the values in the elements of the array iSum are made 'Zero'. Then the values stored in iSum[1] and iSum[2] are copied into the buffer 's' and sent on COM2.

Functions Called:

SendString
ScrollWin
SetCursor
PutRoadInfo

Called by:

ProfileMode3

Example Calls:

PostRut()

Function name: *ProcessFunctions*

Input Parameters:

scan	unsigned char type

Return Value:

None

Global Variables Set and Used:

DisplayMode	Used
	Set
ASCIIMODE	Used
HEXMODE	Used
bMetric	Used
	Set
NMODES	Used
DatacollectMode	Used
szMode	Used
CONSTR	Used
iConstrPreLen	Used
SDK	Used
iStaticAcou	Used
	Set
ACCONLY	Used
ACC_ACOUST	Used
SI_AVGSPD	Used
iRutMethod	Used
CONSTR	Used

Description:

This function performs different functions depending on the value of the variable that is being passed to it namely scan. If the value of the variable 'scan' is 'PGUP' then it calls the routine Upload which sends a file. Similarly if the value of the variable is '0x51' then it asks for a file name then and then it open it and then it is sent using the

sersend function and so on. This function does different activities depending on the value that is being passed as a parameter to this function.

Functions Called:

Upload	FlushPort
serrecv	EscapeHit
sersend	serclose_q
SERCLOSE	SetCursor
CommParam	OpenComm
DisplayCommParam	ScrollWin
readkey	KeypadMode
LoadedCorr	TestMode
ProfileMode2	ProfileMode3
SDProfile1	Mode6
IRI_RUTMode	DisplayHelp

Called by: MonitorMode

Example Calls: ProcessFunction(key.scan);

Function name: *ProfileMode2*

Input Parameters:

acunit	int type
	Set

Return Value:

None

Global Variables Set and Used:

bAbort	Set	fData	Set
iOverrun	Set	iFrame	Set
bDistMode	Set	bCountAcoustic	Set
DataCollectMode	Used	SDK	Used
iConstrPreLen	Set	nInline	Set
fps	Set	mph	Set
mask	Set	CONSTR	Used
str	Set	PortNo	Used

Description: First the filter is set if the filter is not set. It then clears the rows, draws the border and sets the cursor. It then prints in which mode (data collect mode) it is in. Then it gets the name of the file that should be used for storing the collected data. If there is no file name then it means that we are not interested in collecting the data and so the global variable bAbort is set to TRUE. If the global variable bAbort is set to false then it means that we are interested in collecting the data. So the acoustic devices are selected by calling the function 'GetAcoustic' which

gets the numbers of the acoustic devices that should be used for the collection of the acoustic data. Then these numbers that correspond to the acoustic devices should be sent to the siometer. This is done by calling the procedure SetAcUnit. If an escape key has been hit then the collection of data is aborted. Then the file into which the collected data is being written is closed. Then it prints 'ABORTED" on the screen. If escape key is not hit on the screen then it continues for the collection of the data. The siometer is sent 'p' which tells the siometer to collect the data.

The vehicle on which the siometer is mounted and run for some distance before the actual section begins so as to initialize. The section on which the vehicle is run for initialization is called 'presection'. Then at the beginning of the correct section the procedure 'WaitForFirst' is called. The lower order 4 bits are masked. This is done for comparison purpose. The various cases are 'acceleration' , 'speed in mph', 'feet per second' etc. For each of the above we read again from the siometer to know the value. If it is '0xe0' then it means that the reading is in miles per hour. So the next time we read the siometer we get the value of speed in miles per hour. In the default case if we are running the vehicle in a presection, we open the file and print the collected data in a file and 16 values in per line. To abort the collection of the data just press 'escape'. Then the file that is being used for storing the collected data is closed.

Functions Called:

SetFilter	ClearRows	Border
SetCursor	DistanceMode	SetSdMode
GetFileName	GetAcoustic	SetAcUnit
GetSPF	GetDistance	StopCollect
EndMessage	Displaypoints	DisplayStatusForm
SetParam	SendCommand	time
WaitForFirst	CheckTime	MasterOutput
serrecv	kbhit	readkey
CheckComm	SaveOrNot	

Called by:

ProcessFunction

Example Calls:

ProfileMode(0)
ProfileMode(1)

Function name: *ProfileMode3*

Input Parameters: None

Return Value: None

Global Variables Set and Used:

szMode	Used
RutCorr.Mounting	Used
RutCorrloading	Used
iRutMethod	Used
bAbort	Set
	Used
sample	Used
bExtraComm	Used
szActiveAcoustic	Set
fData	Used

Description: This function asks the user to enter the a file name for storing the output. Then it will ask the user to enter two lines of comment. If the variable Comment is set to 'N' then no comment is asked. Then the acoustic devices are selected by calling the function "GetAcoustic'. The user can select the acoustic devices that should be

used to collect the data. The devices are numbered 1 through 5 (left to right). The user can select these by just typing in the numbers. Similarly the distance between samples in feet need to be entered. The distance is also entered followed by 'f' or 'F' for feet or 'm' or 'M' for miles. Then it prompts "Hit any key to start SI". While the data collection is in progress we may hit any number between 0 and 9 to mark Pavement Management Information systems comment codes for that section. Then number will be displayed until the beginning of the next section. When then target distance is reached the collection of data is terminated. Data that is collected is saved automatically. Then user can terminate the program by pressing the 'escape key' on the keyboard in which case it asks the user whether to save the data that was collected till that point or not. The user is prompted to enter 'Y' or 'N' . If the user enters 'Y' then it will be saved else it will be discarded. This is only for the data collection that has been terminated abruptly.

Functions Called:

GetMem	ClearRows
SetCursor	DistanceMode
GetFileName	GetAcoustic
GetSPF	GetDistance
DisplayPoints	DisplayStatusForm
SetExtractRatio	SetAcUnit
InitRut	SetInternalSPF
SendCommand	sersend
FlushPort	WaitForFirst
DistManage	serrecv
WriteSi	ScrollWin
GetRut	readkey
PutRoadInfo	PauseRut
CheckComm	StopCollect
SaveOrNot	CloseRut
EndMessage	

Called by:

ProcessFunction

Example Calls:

ProfileMode3()

Function name: *PutRoadInfo*

Input Parameters:

char c	Used

Return Value:

None

Global Variables Set and Used:

cRoadInfo	Set
	Used

Description: It puts the road information on the screen. It draws the border and the sets the position of the cursor on the screen.

Functions Called:

Border
SetCursor

Called by:

ProfileMode3
PostRut

Example Calls:

PutRoadInfo(key)

Function name: *readkey*

Input Parameters:

None

Return Value:

SCANCODE scancode — The scancode is of type SCANCODE.
where
typedef struct {
unsigned char ascii;
unsigned char scan;
} SCANCODE;

Global Variables Set and Used:

None

Description:

It reads a character from the keyboard buffer. If the buffer does not contain a character, the function waits until a character is entered. Then the character is read and removed from the keyboard buffer. It uses the INTERRUPT 0x16 - Function 0h.
The keyboard buffer stores the ASCII code and then the scancode which is the number of the key that generated the ASCII character.
The ASCII code is stored in scancode.ascii and the scancode is stored in scancode.scan.

Scancode is the number of the key that generated the ascii charecter. (For more info on Scancodes, please refer to Appendix A
The keyboard processor converts the electrical impulse indicating the key position into a number called scancode

Functions Called:

int86 INTERRUPT 0x16h Function 0h

Called by:

CommParam
ProfileMode3
TestMode
WaitForFirst
KeypadMode
LoadedCorr
MonitorMode

Example Calls:

readkey()

Function name: *ReadRutCorr*

Input Parameters:

None

Return Value:

None

Global Variables Set and Used:

RutCorr.Loading	Set
	Used
sRutCorr	Used

Description: It opens a file called "RUTCORR.INI" in read mode. The variable sRutCorr contains the name of this file. It reads file into the array called RutCorr.Loading if the file exits, else the array is filled with zeros.

Functions Called:

None

Called by:

LoadedCorr
Main

Example Calls:

ReadRutCorr()

Function name: *SaveRutCorr*

Input Parameters:

None

Return Value:

None

Global Variables Set and Used:

sRutCorr	Set

Description: This function is used to save the rut correction (that has been read into the array RutCorr.Loading) in to the file whose name is stored in the variable called "sRutCorr" and the file is closed.

Functions Called:

None

Called by:

LoadedCorr

Example Calls:

SaveRutCorr()

Function name: *ScrollWin*

Input Parameters:

int n	Number of lines to scroll Set and Used
int lr	Upper left row Used
int lc	Upper left column Used
int rr	Lower right row Used
int rc	Lower right column Used

Return Value:

None.

Global Variables Set and Used:

None

Description:

It scrolls the window up or down by a specified number of lines. If the value of 'n' greater than 0 then it scrolls the Window up by lines 'n'. It writes blank lines at the bottom of the screen. If the value of 'n' is less than 0 it scrolls the window down by n lines writing n blank lines at the top of the screen.
The register AH is loaded with "6" if it has to scroll up otherwise it will be loaded with "7" if it has to scroll down.
The variables 'lr' and 'lc' correspond to row column of upper left corner and 'rr' and 'rc' correspond to lower right corner. It uses the INTERRUPT 10h - Function 06h for scroll up, and INTERRUPT 10h - Function 07h for scroll down.

Functions Called:

int86	INTERRUPT 0x10h - Function 06h	For Scroll Up
	INTERRUPT 0x10h - Function 07h	For Scroll Down

Called by:

ProcessFunctions
PostRut
ProfileMode3
IRI_RUTMode

Example Calls:

ScrollWin(1,0,0,23,79)

Function name: *SelectAcoustic*

Input Parameters:

s	char array
	Used
acunit	Pointer to int
	Set

Return Value:

Integer

Global Variables Set and Used:

szActiveAcoustic	Set
AcousticMask	Set
DataCollectMode	Used
SDK	Used
CONSTR	Used
bDistMode	Used

Description: The function SelectAcoustic is used to select the acoustic data. If the contents of the array are between value 1 and value 6 then they are copied into szActiveAcoustic. These represent the numbers of the acoustic devices. Else they are discarded.

Functions Called:

None

Called by:

TestMode
GetAcoustic

Example Calls:

SelectAcoustic(str, acunit)
SelectAcoustic("12345", &acunit)

Function name: *SendCommand*

Input:

char *s	Set
	Used

Return Value:

None.

Global Variables Set and Used:

bAbort	Set
PAUSE	Used
PortNo	Used

Description:

This function sends the given character string on the given port number.
First copy the string to be send into space pointed by variable called command. The pointer to character "s" is also set to command.

Send one character to the given port no. Wait for the same character to come back on the same port. If the character comes back within a given time limit then go ahead and send the next character in the string. If the character does not come back we need to send the same character over again.

When the character sent over to a port does not come back after a given amount of time,
the character 0x08 is sent over to the port and then the port is flushed and then the same character is sent over again. The character 0x08 is the back space character. If a wrong character was sent it needs to be erased from the port.
After the full string has been sent, the carriage return character is sent over the same port.

Functions Called:

FlushPort, EscapeHit, sersend and WaitChar.

Called by:

ProfileMode2, SetFilter, SetExtractRatio, GetMem, ProfileMode3, SetSdMode, SetMem, SetAcUnit, DistanceMode, IRI_RUTMode, SDProfile, KeyPadMode and TestMode.

Example Calls:

SendCommand(s), SendCommand(sVal).

Function name: *SendString*

Input Parameters:

int iP	Port number on which a string is sent.
	Used
char *s	String that needed to be sent.
	Set
	Used

Return Value:

None

Global Variables Set and Used:

None

Description:

This function is used to send a string pointed by the character pointer *s on the port whose number iP.

Functions Called:

sersend

Called by:

WriteSI
PostRut

Example Calls:

SendString(COM2, s)

Function name: *SerialPortStatus*

Input Parameters:

PortNo Port number.
Used

Return Value:

The status of the port which is a unsigned integer.

Global Variables Set and Used:

PortNo Used.

Description:

This function returns the status of the serial port whose number is PortNo. It uses the INTERRUPT 14h - Function 03h. For this AH is 03 and DX is the port number.

Functions Called:

int86 INTERRUPT 0x14h - Function 03h.

Called by:

CheckComm.

Example Calls:

SerialPortStatus(int PortNo).

Function name: *SetAcUnit*

Input Parameters:

acunit	Int type
	Used

Return Value:

None

Global Variables Set and Used:

PortNo	Used
AcousticMask	Used

Description: This function copies the acunit into the buffer 's' along with "S 4E16". Then this is sent to siometer using the SendCommand function. Then "S 4E58" is sent to siometer to mask the siometer. Then the siometer is read. Then AcousticMask and odd byte of the word value is copied into the buffer and sent to siometer. The acoustic mask contains the acoustic device numbers that have been selected for the collection of the data. Then escape character is sent to indicate the completion of sending.

Functions Called:

SendCommand
sersend
FlushPort
GetString

Called by:

IRI_RUTMode
ProfileMode2
ProfileMode3
Mode6
SDProfile1

Example Calls:

SetAcUnit(acunit)

Function name: *SetCursor*

Input Parameters: r and c

r	Row
	Used
c	Column
	Used

Return Value:

None

Global variables set and used

None

Description: Positions the cursor on the screen on the specified page number. It positions the cursor on the screen where the row number is 'r' and column number is 'c'. It uses INTERRUPT10h- Function 02h.

For this AH = 02, BH is the page number , DH is row (00h is the most top row) and DL is column(00h is left most column of the page).

Functions Called:

int86 INTERRUPT10h- Function 02h

Called by:

Mode6
DistanceMode
ProfileMode3
DisplayPoints
PauseRut
Border
PostRut
GetAcoustic
KeypadMode
GetFileName
EndMessage
PutRoadInfo
CheckComm
IRI_RUTMode
SaveOrNot
GetSPF
GetDistance
ProcessFunctions
DispalyStatusForm
ProfileMode2
TestMode
LoadedCorr
MonitorMode
SDProfile
WaitForFirst

Example Call:

SetCursor(23, 0)

Function name: *SetExtractRatio*

Input Parameters:

er int type

Return Value:

None

Global Variables Set and Used:

None

Description:

This function copies " S 4E50" along with the value of er in to the buffer 's'. Then the contents are sent to the siometer.

Functions Called:

SendCommnad

Called by:

Example Calls:

Function name: *SetFilter*

Input Parameters:

b	unsigned char type Used

Return Value:

None

Global Variables Set and Used:

None

Description: It sends 'F' using SendCommand and pauses for 100 milliseconds by calling the Delay routine. The value in character 'b' is copied into the buffer 'str' and it is sent to the siometer using the SendCommand. Then it prints "Filter set to " on the screen. This acts as a low pass filter which passes up to a particular value and lower.

Functions Called:

Delay
SendCommand

Called by:

ProfileMode2

Example Calls:

SetFilter(0x2c)

Function name: *SetInternalSPF*

Input Parameters:

iSioSPF	An integer Used

Return Value:

None

Global Variables Set and Used:

None

Description: This function sends "A" first to the siometer. Then it waits or pauses for 100 milliseconds. Then it copies the iSioSPF in to the buffer 's' and this buffer is sent to the siometer.

Functions Called:

SendCommand
Delay

Called by:

SetParam
ProfileM0ode3

Example Calls:

SetInternalSPF(iInternalSPF)

Function name: *SetMem*

Input Parameters:

long lAddr	This contains an address Used
int iVal	It is also an address (*) . Example values are 0x100, 1 etc. Used
BOOL bWord	This can have the value of 0|1 cause it is BOOLEAN. Generally used are CHANGEWORD = 1 CHANGEBYTE = 0 Used

Return Value:

None

Global Variables Set and Used:

PortNo	Used

Description:

This function uses the input parameters in the following manner. It uses the value of the input parameter lAddr which is anded with 0xfffffffeL to ensure that the last bit is set to 0 and the number is even. The result is copied to the "s" buffer using sprintf . The value of iVal is copied into "sval" buffer using sprintf. The string contained in buffer "s" is sent to PortNo using the function SendCommand. The string is obtained using GetString function.
If bWord = CHANGEBYTE then it changes the odd byte and keeps the even byte. If bWord = CHANGEWORD then it changes the even byte and keeps the odd byte.
It sends the value of "sVal" to the PortNo using the SendCommand. It then sends the escape character and flushes the port.

Functions Called:

SendCommand
FlushPort
sersend
GetString.

Called by:

SDProfile
IRI_RUTMode.

Example Calls:

SetMem(long lAddr, int iVal, BOOL bWord)

Function name: *SetParam*

Input Parameters:

sample	double type Used
speed	double type

Return Value:

None

Global Variables Set and Used:

None

Description:

This function sets the parameters iInternalSPf and iExtractratio and then sends them to the siometer by calling the procedures SetInternalSPF and SetExtractRatio

Functions Called:

SetInternalSPF
SetExtractRatio

Called by:

Example Calls:

Function name: *SetSdMode*

Input Parameters:

DataCollectMode	int type

Return Value:

None

Global Variables Set and Used:

SDK	Used
ODD_START	Used
PortNo	Used

Description:

This function sends the siometer first "S 10510A". Then it reads the buffer using the function GetString. IF the

DataCollectMode is SouthDakota Mode then the variable 'cFlag' is set to 1 else it is set to 0. Then the cFlag value along with the buffer contents from the 11th cell to the end of the buffer. All these are copied into the buffer 's' and sent to the siometer on the port number 27 and the port is flushed.

Functions Called:

SendCommand
GetString
sersend
FlushPort

Called by:

Example Calls:

Function name: *StopCollect*

Input Parameters:

None.

Return Value:

None

Global Variables Set and Used:

None

Description:

It sends an Escape character and then flushes the port and once again sends the Escape charecter on the port referred by the PortNo.

Functions Called:

sersend
FlushPort.

Called by:

ProfileMode3
ProfileMode2
IRI_RUTMode
SDProfile.

Example Calls:

StopCollect()

Function name: *TestMode*

Input Parameters:

iMode	int type
Avg	floating point array of five elements

Return Value:

None

Global Variables Set and Used:

DatacollectMode	Set
	Used
ACC_ACOUST	Used
bDistMode	Set
	Used
bAbort	Set
	Used
TM_TEST	Used
acunit	Used
bCountAcoustic	Used
PortNo	Used
mph	Set
fps	Set

Description: The variable bAbort is set to FALSE if execution is not to be aborted. If it is so then rows are cleared and cursor is set by calling the procedures Clear Rows and SetCursor. Then it prints "Testing.. " on the screen. 'Z' is sent to the siometer and pauses for 100 millisecond by calling Delay function. Then it sends "25" on the siometer and flushes the port. Then bit number 6 is made sure to set to 1 by bitwise 'OR'ing with 0x40 which is represented in the binary as 100000. The truth table of bitwise 'OR' is

TRUTH TABLE

A	B	TruthValue
0	0	0
1	0	1
0	1	1
1	1	1

Sample variable, which is the number of samples is set to 1. Then by calling the SendCommand character "p" is sent to the siometer. Then the four lower order bits are set to '0' by bitwise 'AND'ing with 0xf0. When a byte is received from the siometer then the lower 4 bits are masked and the Most significant 4 bits are kept in the variable called mask. If the value in the variable is equal to the hexadecimal value 80 then the original value received from the siometer is shifted to left by eight bits introducing 8 zeros in the least significant bits position and this value is stored in variable called wVal. Then the siometer is read again and the received value added to the value stored in wVal. This is done until the value that has been received is not 'ffff'. Then the byte that has been received from the siometer is printed on the screen. If the escape key has been hit then the variable 'Comm_err' is set to 1. Then '27' is sent to the siometer to indicate that communication has been aborted. If any of the variables Comm_err or iOverrun or iFrame is set then the comment 'Communication Error ! Try again' is printed on the screen, If bDistMode is set then the comment 'Set to Time mode first' is printed on the screen.

Functions Called:

ClearRows	Border
SetCursor	SetSdMode
SendCommand	SetAcUnit
SelectAcoustic	WaitForFirst

serrecv	CheckComm
EscapeHit	sersend
Delay	FlushPort

Called by:

Loaded Corr
ProcessFunction

Example Calls:

TestMode(TM_TEST,AvgDisp)

Function name: *Upload*

Input Parameters:

fname	Name of the file.
	Used

Return Value:

None.

Global Variables Set and Used:

PortNo	Used.

Description:

This function sends the complete file on the port that is specified in PortNo. We can abort sending the file at any time by pressing the Escape key on the keyboard.

Functions Called:

EscapeHit
sersend
serrecv
FlushPort

Called by:

ProfileMode3

Example Calls:

WriteSI(sival, wsv);

Function name: *WaitChar*

Input Parameters:

int Port No	Port Number
	Used
long ms	Amount of time the function has to wait for a character to come in on the port
	Used

Return Value:

unsigned wTemp The character obtained from the specified port.

Global variables set and used:

None

Description:

A port is specified from where we need to receive a character. We keep on waiting on the port for the specified amount of time (variable ms) until a character comes in.

Functions Called:

serrecv
clock

Called by:

SendCommand, GetString

Example Call:

WaitChar(PortNo, 500L)

Function name: *WaitForFirst*

Input Parameters:

unsigned	int *wFirst	The address of character obtained from keyboard set
BOOL bDisp		printf statements are displayed if it is set used

Return Value:

TRUE| FALSE

Global Variables Set and Used:

PortNo Used.

Description:

If bDisp is true then it prints " Waiting for the first point. Hit any key to abort".
It waits for a character to be read from the PortNo. If a valid character comes in or if the user hits any key to abort, the routine comes out of the do loop which is waiting for the character.
The pointer to an unsigned integer is assigned
*wFirst = wTemp
If bDisp is true then some rows are cleared.
If the keyboard is hit again then the statement " Function Aborted.....Hit [ESC] " is printed and another key is read and FALSE is returned , otherwise TRUE is returned.

Functions Called:

readkey
kbhit

SetCursor
serrecv
ClearRows

Called by:

TestMode
ProfileMode3
ProfileMode2
IRI_RUTMode
SDProfile1

Example Calls:

WaitForFirst(&wTemp, TRUE)
WaitForFirst(&wTemp, FALSE)

Function name: *WriteCh*

Input Parameters:

char ch	Character to be displayed on the screen
	Used
char attr	Attribute of the character
	Used

Return Value:

None

Global Variables Set and Used:

None

Description:

This function writes a specified character and attribute to display at the current cursor position. The value of 'regs.x.cx' represents the number of times the character to be displayed. It uses interrupt 10 , function 09.

Functions Called:

int86 INTERRUPT 0x10h - Function 09

Called by:

Border

Example Calls:

WriteCh(186, 7), WriteCh(205, 7).

Function name: *WriteSI*

Input Parameters:

si unsigned int type

	Used
wsv	unsigned long type.
	Used

Return Value:

None.

Global Variables Set and Used:

fSI	Used.
fWSV	Used.
iSILine	Set.
iSICnt	Used and Set.
bExtraComm	Used.
COM2	Used.

Description:

fSI and fWSV are file pointers to the files namely RTRIDE.DAT and RTWSV.DAT respectively. This function prints 10 SI's per line including the line number into the file RTRIDE.DAT. Similarly it prints 10 WSV's per line including the line number into file named RTWSV.DAT.

Functions Called:

SendString.

Called by:

ProfileMode3.

Example Calls:

WriteSI(sival, wsv)

DOCUMENTATION FOR FUNCTIONS IN OTHER FUNCTIONS FOR TALK.EXE 78

Function name: serinit_q(port, config, irq) 78

Function name: intserv 78

Function name: serclose_q(port no, irq) 79

Function name: sersend(port, char) 80

Function name: serrecv(port) 80

Function name: CheckTime 23

Function name: Get_Header(char *Filename)

Function name: DispCorr 26

Function name: FlushPort 34

Function name: CheckComm 22

Function name: CloseRut 24

Function name: DisplayHelp 28

Function name: DisplayPoints 28

Function name: WaitChar 72

Function name: SendCommand 63

Function name: GetString 45

Function name: DistanceMode 30

Function name: DistManage 31

Function name: Delay

Function name: EscapeHit

Function name: SetCursor

Function name: GetCursor

Function name: GetIRQ

Function name: SaveOrNot

Function name: ScrollWin

Function name: WriteCh

Function name: Border

Function name: CommParam 25

Function name: DisplayStatusForm

Function name: StopCollect

Function name: WaitForFirst

Function name: GetMem

Function name: SetMem

Function name: EndMessage

Function name:SerialPortStatus

Function name: MonitorMode

Function name: WriteSI

Function name: Upload

Function name: readkey

Documentation for functions in other Functions for talk.exe

Function name: *serinit_q(port, config, irq)*

Input:

port Port Number to be accessed when the given (irq) interrupt comes in
config config is a byte. config contains data given below which is composed in a byte.
int PortNo, int BaudRate, int Parity
int DataBits, int StopBits

irq Port driver to be initiated by the (irq) level.

Output:

None

Global variables set and used:

_bufaddr, _bufin, _bufout, _intsav

Description:

This procedure should have the port number, irq number and config as input. This procedure can only initialize COM1, COM2 and COM3. You can setup the interrupt for COM3 by giving it the initialization files talk.ini. This procedure sets up the communication parameters for the port like PortNo, BaudRate, Parity, DataBits and StopBits- this information is given in config. This procedure also initializes the circular buffer where we can write incoming data for the port. It also sets up another routine (Interrupt Service routine)
for the given interrupt which initiates the port driver.
COM1 uses IRQ4
COM2 uses IRQ3
and COM3 can be setup for IRQ2 or IRQ5 (*)

Functions Called:

intserv (asm , serqs.asm)

Called by:

OpenComm

Example Call:

serinit_q(PortNo, serconfig, GetIRQ(PortNo));

Function name: *intserv*

Input:

None

Output:

None

Global variables set and used:

Not sure

Description:

This routine is the ISR routine which is called when the port needs to be accessed. This routine is used by the serinit_q procedure. This routine is called when the interrupts set up for COM1, COM2 and COM3 interrupt the processor. This routine calls the ports driver for the respective ports set up for the interrupts.

Functions Called:

putb

Called by:

serinit_q (asm, serqs.asm)

Function name: *serclose_q(port no, irq)*

Input:

port The port to be closed
irq The interrupt used for the port given above

Output

Global variables set and used: _intsav

Description:

This is used for closing the serial port. Here the interrupt is specified for the port. This routine can be used only after serinit_q has been used for initialize the same port for the given interrupt. You must have noticed that, in serinit_q the interrupt service routine (ISR) for the given interrupt was changed to another ISR which calls the port driver of the given port. After we close the port we need to change the interrupt service routine of the interrupt back to the old interrupt service routine. The address of the old interrupt service routine has already been saved previously. The interrupt is disabled during the change of the ISR.

Functions Called:

None

Called by:

MonitorMode (C, bt.c)
ProcessFunctions (C, talk70.c)

Example Call

serclose_q(PortNo,GetIRQ(PortNo));

Function name:*sersend(port, char)*

Input:

port
char

Output:

None

Global variables set and used:

None

Description:

This routine is used to send a byte of data on a given port number.

Functions Called:

None

Called by:

Process Functions (C, talk70.c)
Upload (C, bt.c)
KeyPadMode (C, talk70.c)
WaitChar (C, bt.c)
SetSdMode (C, talk70.c)
SetAcUnit (C, talk70.c)
TestMode (C, talk70.c)
LoadedCorr (C, talk70.c)
DistanceMode (C, talk70.c)
StopCollect (C, bt.c)
EndMesage (C, bt.c)
ProfileMode3 (C, mode3.c)
DistManage (C, mode3.c)
SendString (C, rtrut.c)
SdProfile1 (C, sd1.c)
IRI_RutMode (C, mode7.c)
MonitorMode (C, bt.c)

Example Calls

sersend(PortNo, Char)

Function name: *serrecv(port)*

Input:

port Port number on which a data byte is received

Output:

None

Global variables set and used:

None

Description:

This is used to receive data from the serial port. This procedure receives a byte of data. If the value returned is FFFF, then the buffer is empty.

Functions Called:

getb(asm, serqs.asm)

Called by:

Process Functions (C, talk70.c)
Upload (C, bt.c)
KeyPadMode (C, talk70.c)
WaitChar (C, bt.c)
SetSdMode (C, talk70.c)
SetAcUnit (C, talk70.c)
TestMode (C, talk70.c)
ProfileMode3 (C, mode3.c)
SdProfile1 (C, sd1.c)
IRI_RutMode (C, mode7.c)
MonitorMode (C, bt.c)
WaitForFirst (C, bt.c)
ProfileMode2 (C, talk70.c)
Mode6 (C, mode6.c)
GetFloat (C, mode7.c)
GetMsg (C, mode7.c)

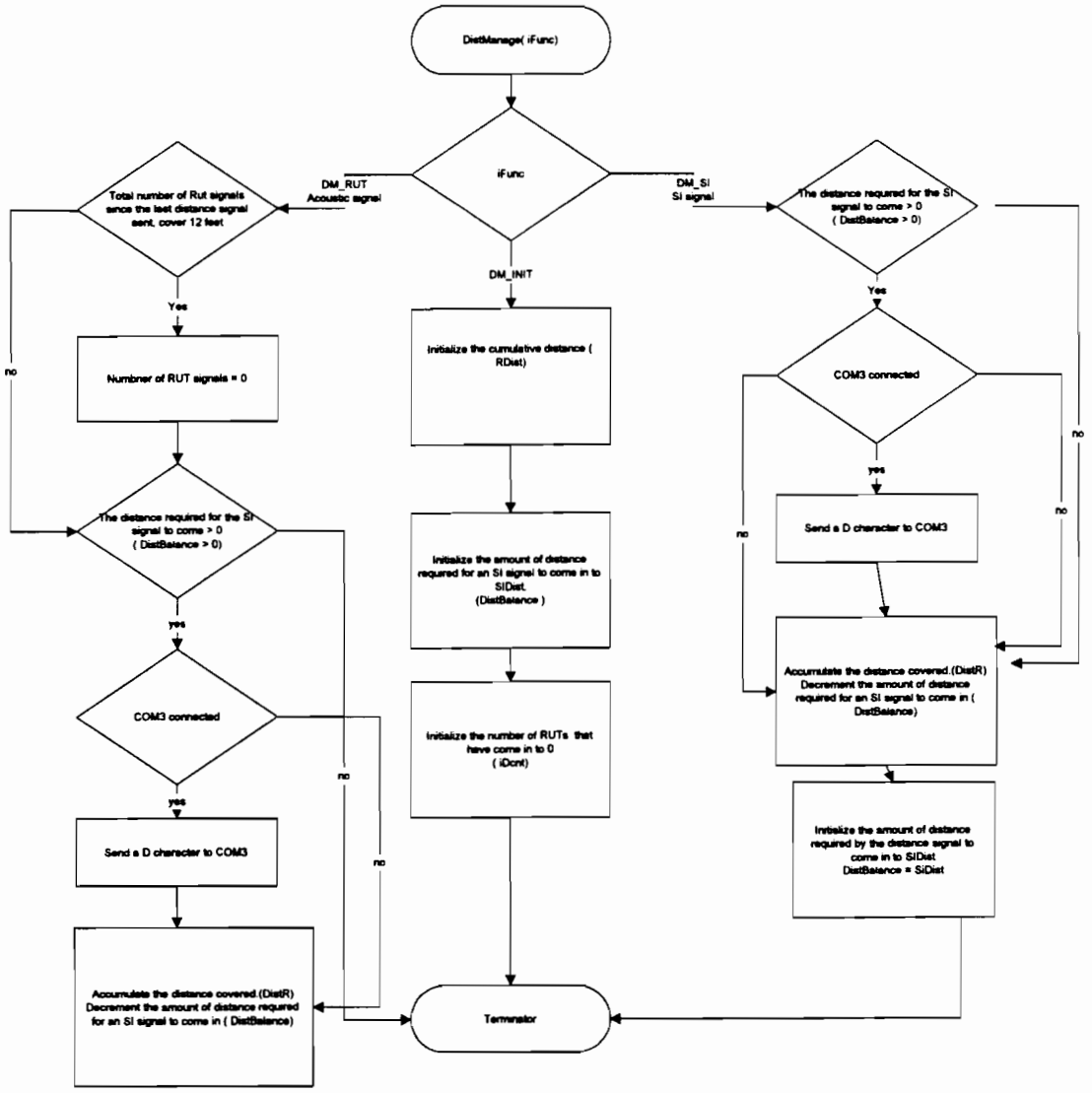
Example Calls

serrecv(PortNo)

Tips:

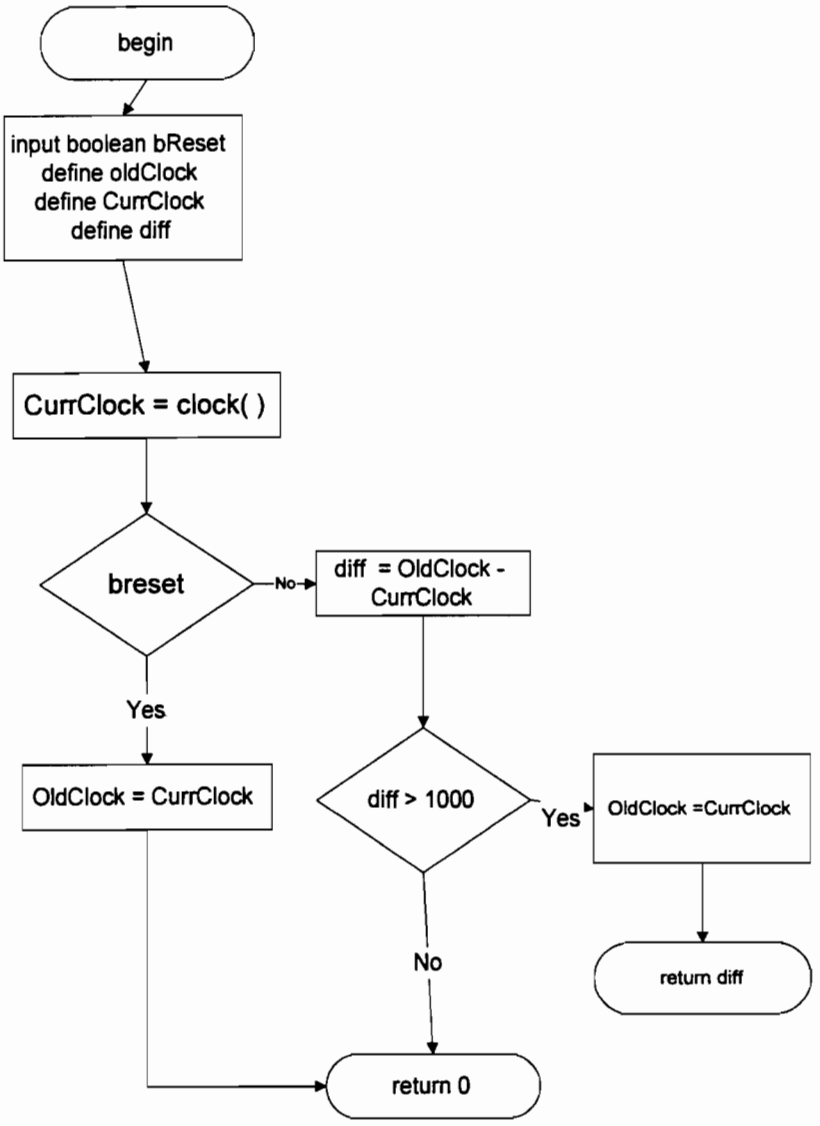
ISR - Interrupt service routine.
You need to have knowledge of ISR and how the ISR can be changed for an interrupt.

FlowCharts

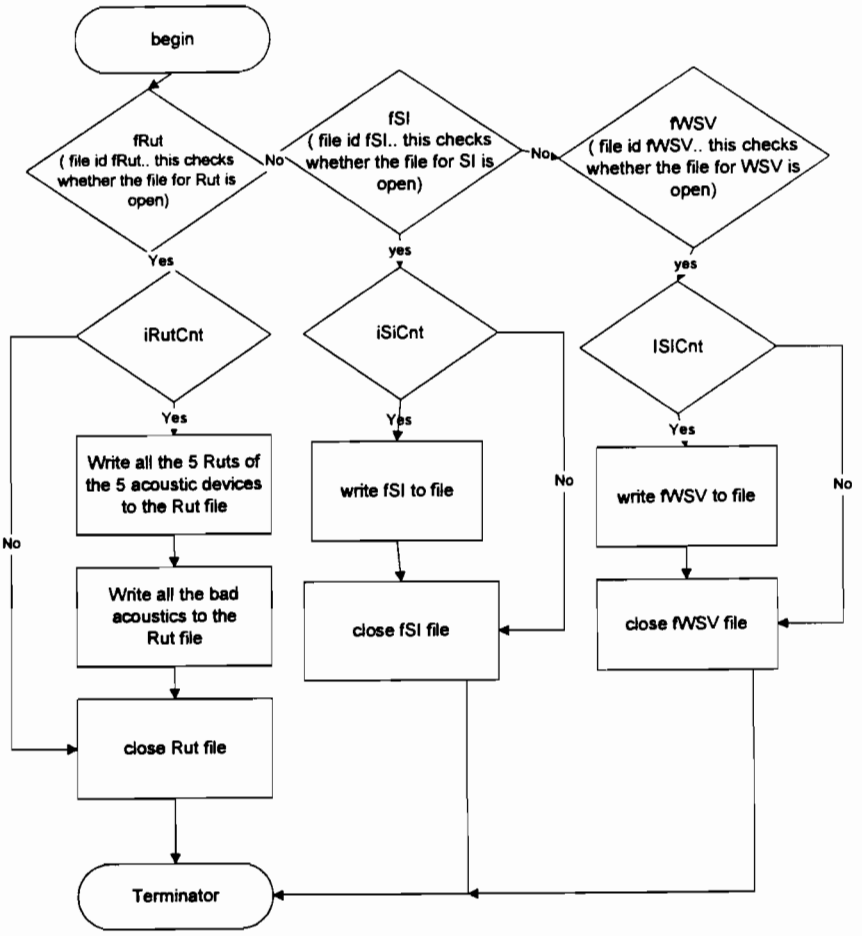


FlowChart for DistManage

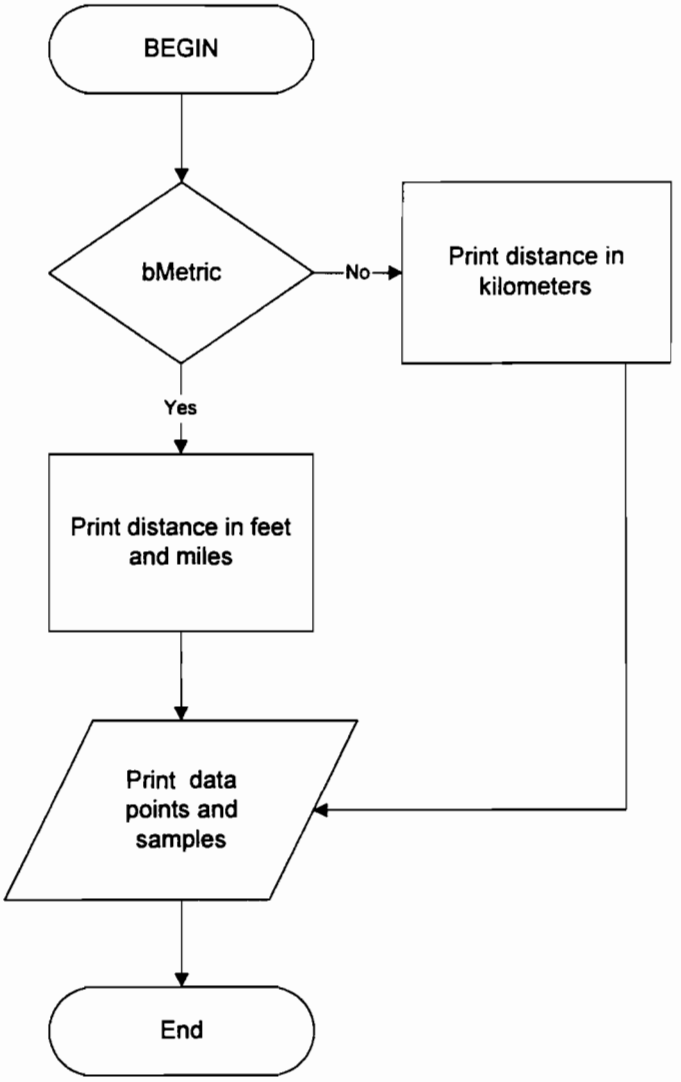
CheckTime


begin
input boolean bReset
define oldClock
define CurrClock
define diff
CurrClock = clock( )
breset
No
diff = OldClock - CurrClock
Yes
OldClock = CurrClock
diff > 1000
Yes
OldClock =CurrClock
return diff
No
return 0

CloseRut


begin
fRut
(file id fRut.. this checks whether the file for Rut is open)
No
Yes
fSI
(file id fSI.. this checks whether the file for SI is open)
No
yes
fWSV
(file id fWSV.. this checks whether the file for WSV is open)
yes
iRutCnt
Yes
No
iSiCnt
Yes
No
ISICnt
Yes
No
Write all the 5 Ruts of the 5 acoustic devices to the Rut file
Write all the bad acoustics to the Rut file
close Rut file
write fSI to file
close fSI file
write fWSV to file
close fWSV file
Terminator

DisplayPoints



List of Functions Sorted

Sorted.doc:
This file gives the list of functions. The file where the functions occur are specified in this file . The notation [terminal] indicates that the function does not call any more user-defined functions.

Border(int lr, int lc, int rr, int rc) -bt.c
CheckComm()- bt.c
CheckTime(BOOL bReset) - talk70.c [terminal]
CloseRut() - rtrut.c [terminal]
CommParam() - bt.c
COUNT_TO_INCH(i) - rtrut.c[terminal]
Delay(long ms) - bt.c [terminal]
DispCorr() - talk70.c [terminal]
DisplayCommParam()- bt.c[terminal]
DisplayHelp() - talk70.c
DisplayPoints(int iRow, long nPts, double dist, double sample) - talk70.c
DisplayStatusForm(int iRow) - bt.c
DistanceMode(BOOL bDisplay) - talk70.c
DistManage(int iFunc) - mode3.c
EndMessage()- bt.c
EscapeHit()- bt.c
FlushPort(int PortNo) - bt.c [terminal]
GetAcoustic(int iRow, int *acunit) - talk70.c
GetCursor(int *r, int *c) - bt.c [terminal]
GetDistance(int iRow, char *unit, double *dist) - talk70.c
GetFileName(int iRow) - talk70.c
GetFloat() - mode7.c
GetHeader(char *FileName) - talk70.c [terminal]
GetInternalSPF() - talk70.c
GetIRQ(int PortNo) - bt.c [terminal]
GetMem(long lAddr) - bt.c
GetMsg(char *s) - mode7.c
GetRut() - rtrut.c
GetSCKPMI() - talk70.c
GetShort() - mode7.c
GetSPF(int iRow, int Mode, FILE *f) - talk70.c
GetString(char *s, int iLen) - bt.c
InitRut() - rtrut.c [terminal]
InitSI(char *fn) - talk70.c
IRI_RUTMode() - mode7.c
KeyPadMode() - talk70.c
LoadedCorr() - talk70.c
main(int argc, char *argv[]) - talk70.c
MasterOutput(char szFormat[], unsigned wVal, BOOL bPreSection) - talk70.c
Mode6() - mode6.c
MonitorMode()- bt.c
OpenComm(int PortNo, int BaudRate, int Parity, int DataBits, int StopBits) - bt.c
PauseRut() - rtrut.c
PostRut() - rtrut.c
ProcessFunctions(unsigned char scan) - talk70.c
ProfileMode2(int acunit) - talk70.c
ProfileMode3() - mode3.c
PutRoadInfo(char c) - mode3.c
readkey() - bt.c [terminal]

ReadRutCorr() - talk70.c
SaveOrNot()- bt.c [terminal]
SaveRutCorr() - talk70.c
ScrollWin(int n, int lr, int lc, int rr, int rc) - bt.c [terminal]
SDProfile1(int acunit) - sd1.c
SelectAcoustic(char s[], int *acunit) - talk70.c [terminal]
SendCommand(char *s) - bt.c [terminal]
SendString(int iP, char *s) - rtrut.c
SerialPortStatus(int PortNo) - bt.c
SetAcUnit(int acunit) - talk70.c
SetCursor(int r, int c) - bt.c [terminal]
SetExtractRatio(int er) - talk70.c
SetFilter(unsigned char b) - talk70.c
SetInternalSPF(int iSioSPF) - talk70.c
SetMem(long lAddr, int iVal, BOOL bWord) - bt.c
SetParam(double sample, double speed) - talk70.c
SetSdMode(int DataCollectMode) - talk70.c
StopCollect()- bt.c
TestMode(int iMode, float Avg[5]) - talk70.c
Upload(char *fname) - bt.c
WaitChar(int PortNo, long ms){ //[60] - bt.c
WaitForFirst(unsigned int *wFirst, BOOL bDisp) //[60] - bt.c
WriteCh(char ch, char attr) - bt.c
WriteSI(unsigned int si, unsigned long wsv) - rtrut.c [terminal]
serinit_q - serqs.asm [terminal]
intserv - serqs.asm [terminal]
serclose_q - serqs.asm [terminal]
sersend - serqs.asm [terminal]
serrecv - serqs.asm [terminal]

List of Functions for Each File

List of Functions in bt.c / Filename: bt.doc

int GetIRQ(int PortNo)

SCANCODE readkey()

int OpenComm(int PortNo, int BaudRate, int Parity,

int DataBits, int StopBits)

unsigned int SerialPortStatus(int PortNo)

void CommParam()

void DisplayCommParam()

void SendCommand(char *s)

unsigned int WaitChar(int PortNo, long ms){ //[60]

void ScrollWin(int n, int lr, int lc, int rr, int rc)

void SetCursor(int r, int c)

void GetCursor(int *r, int *c)

void WriteCh(char ch, char attr)

void Border(int lr, int lc, int rr, int rc)

void FlushPort(int PortNo)

void MonitorMode()

void Delay(long ms)

void GetString(char *s, int iLen)

void DisplayStatusForm(int iRow)

void CheckComm()

void EndMessage()

void StopCollect()

void SaveOrNot()

BOOL WaitForFirst(unsigned int *wFirst, BOOL bDisp) //[60]

BOOL EscapeHit()

void Upload(char *fname)

void SetMem(long lAddr, int iVal, BOOL bWord)

int GetMem(long lAddr)

Functions related to file bt.c for talk70.exe

1. Function name: *GetIRQ*
   Calls: None
2. Function name: *readkey*
   Calls: int86
3. Function name: *OpenComm*
   Calls: *serinit_q
   *serinit
   outp
   inp

4. Function name: *SerialPortStatus*
   Calls: int86

5. Function name: *CommParam*
   Calls: printf
   *readkey
6. Function name: *DisplayCommParam*
   Calls: printf
7. Function name: *SendCommand*
   Calls: strcpy
   *FlushPort
   printf
   *EscapeHit
   *sersend
   *WaitChar
8. Function name: *WaitChar*
   Calls: *clock
   *serrecv
9. Function name: *ScrollWin*
   Calls: int86
10. Function name: *SetCursor*
    Calls: int86
11. Function name: *GetCursor*
    Calls: int86
12. Function name: *WriteCh*
    Calls: int86
13. Function name: *Border*
    Calls: *SetCursor
    *WriteCh
14. Function name: *FlushPort*
    Calls: printf
    *serrecv
15. Function name: *MonitorMode*
    Calls: *SetCursor
    *OpenComm
    kbhit
    *sersend
    *ProcessFunction
    *ScrollWin
    *serrecv
    putch
    printf
    *serclose_q

*serclose
*readkey

16. Function name: *Delay*
Calls: *clock

17. Function name: *GetString*
Calls: *WaitChar
*EscapeHit
*FlushPort

18. Function name: *DisplayStatusForm*
Calls: *ClearRows
*SetCursor
printf

19. Function name: *CheckComm*
Calls: *SerialPortStatus
*SetCursor
printf

20. Function name: *EndMessage*
Calls: *sersend
*SetCursor
printf
*Border
*readkey
*ClearRows

21. Function name: *StopCollect*
Calls: *sersend
*FlushPort

22. Function name: *SaveOrNot*
Calls: *ClearRows
*SetCursor
printf
gets
strtok
remove

23 Function name: *WaitForFirst*
Calls: kbhit
*readkey
*SetCursor
printf
*serrecv
*ClearRows

24 Function name: *EscapeHit*
Calls: kbhit

25. Function name: *Upload*
Calls: printf
gets
strcpy
fopen
*EscapeHit
fgetc
feof
*sersend
*serrecv
putch
fclose
*FlushPort

26. Function name: *SetMem*
    Calls: sprintf
        *SendCommand
        *GetString
        *sersend
        *FlushPort
27. Function name: *GetMem*
    Calls: sprintf
        *SendCommand
        *GetString
        sscanf

Filename: mode3.doc \ List of functions in mode3.c

DISTTYPE DistManage(int iFunc)
void ProfileMode3()
void PutRoadInfo(char c)

List of functions for mode3.c for talk70.exe

1. Function name: *DistManage*
    Calls: *sersend
2. Function name: *ProfileMode3*
    Calls: *GetMem
        *ClearRows
        *Border
        *SetCursor
        printf
        *GetFilename
        *GetAcoustic
        *GetSPF
        *GetDistance
        *DisplayPoints
        *DisplayStatusForm
        *SetExtractRatio
        *SetAcUnit
        *InitRut
        *SetInternalSPF
        fputc
        *SendCommand
        *sersend
        *FlushPort
        kbhit
        *readkey
        *WaitForFirst
        *time
        *ScrollWin
        *DistManage
        *serrecv
        fputc
        putchar
        *WriteSI
        *GetRut

*PostRut
*PutRoadInfo
*PauseRut
*CheckComm
*StopCollect
fclose
*SaveOrNot
*CloseRut
*EndMessage

3. Function name: *PutRoadInfo*
Calls: *Border
*SetCursor
putchar

List of functions in mode6.c
Filename: mode6.doc

void Mode6()

Functions related in mode6.c for talk 70.exe

Function name: *Mode6*
Calls: *ClearRows
*Border
*SetCursor
strlen
printf
*DistanceMode
*SetSdMode
*GetFileName
*SetAcUnit
*GetSPF
*GetDistance
*StopCollect
fclose
*EndMessage
*DisplayPoints
*DisplayStatusForm
*SetParam
*SendCommand
time
*WaitForFirst
*serrecv
*MasterOutput
fprintf
kbhit
putchar
*CheckComm
*SaveOrNot

List of functions for mode7.c for talk70.exe

1. Function name: *IRI_RUTMode*
   Calls: *SendCommand
   *SetMem
2. Function name: *IRI_RUTMode*
   Calls: *ClearRows
   *Border
   *SetCursor
   printf
   *GetFilename
   *GetAcoustic
   *SetAcUnit
   *SendCommand
   *SetMem
   *GetSPF
   *GetDistance
   *StopCollect
   *EndMessage
   fclose
   *DisplayPoints
   *DisplayStatusForm
   *SetParam
   *time
   *WaitForFirst
   *GetFloat
   *ScrollWin
   *GetMsg
   kbhit
   *sersend
   *serrecv
   *CheckComm
   *SaveOrNot
3. Function name: *GetFloat*
   Calls: *serrecv
4. Function name: *GetShort*
   Calls: *serrecv
5. Function name: *GetMsg*
   Calls: *serrecv

List of functions in rtrut.c
Filename: rtrut.doc

void InitRut()
void CloseRut()
COUNT_TO_INCH(i)
void GetRut()
void PauseRut()
void PostRut()
void WriteSI(unsigned int si, unsigned long wsv)
void SendString(int iP, char *s)

List of related functions for rtrut.c for talk70.exe

1. Function name: *InitRUT*
   Calls: fopen
   fprintf
   strcpy
   sprintf
2. Function name: *CloseRUT*
   Calls: fprintf
   fclose
3. Function name:*COUNT_TO_INCH*
   Calls: None
4. Function name: *GetRUT*
   Calls: *COUNT_TO_INCH
5. Function name: *PauseRUT*
   Calls: *SetCursor
   printf
   *Border
6. Function name: *PostRut*
   Calls: fopen
   fprintf
   sprintf
   *SendString
   *ScrollWin
   *SetCursor
   putchar
   *PutRoadInfo
   printf
7. Function name: *WriteSI*
   Calls: fprintf
   sprintf
8. Function name: *SendString*
   Calls: *sersend

List of functions in sd1.c
Filename: sd1.doc

void SDProfile1(int acunit)

List of related functions for sd1.c for talk70.exe

1. Function name: *SDProfile*
   Calls: *ClearRows
   *Border
   *SetCursor
   printf
   *SetSdMode

*SetMem
*GetFileName
*GetAcoustic
*SetAcUnit
*GetSPF
*StopCollect
fclose
*EndMessage
*DisplayPoints
*DisplayStatusForm
*SetParam
*SendCommand
*WaitForFirst
*serrecv
fprintf
*time
printf
kbhit
*sersend
*CheckComm
*StopCollect
fclose
*SaveOrNot
*EndMessage

List of functions in talk70.c
Filename: talk70.doc

1. List of Functions in talk70.c
void InitSI(char *fn)
char *GetHeader(char *FileName)
main(int argc, char *argv[])
void ProcessFunctions(unsigned char scan)
void DisplayHelp()
int DistanceMode(BOOL bDisplay)
int GetInternalSPF()
int GetSCKPMI()
void SetInternalSPF(int iSioSPF)
void SetAcUnit(int acunit)
void SetSdMode(int DataCollectMode)
void SetExtractRatio(int er)
void SetParam(double sample, double speed)
void GetAcoustic(int iRow, int *acunit)
int SelectAcoustic(char s[], int *acunit)
FILE *GetFileName(int iRow)
double GetSPF(int iRow, int Mode, FILE *f)
void GetDistance(int iRow, char *unit, double *dist)
void DisplayPoints(int iRow, long nPts, double dist, double sample)
void ProfileMode2(int acunit)
void MasterOutput(char szFormat[], unsigned wVal, BOOL bPreSection)
unsigned CheckTime(BOOL bReset)
void KeyPadMode()

void TestMode(int iMode, float Avg[5])
void SetFilter(unsigned char b)
void ReadRutCorr()
void SaveRutCorr()
void DispCorr()

Related functions for talk70.c

1. Functions name: *InitSI(char *fn)*
   Calls : fopen
   fgets
   feof
   strupr
   strtok
   strcmp
   strncpy
   atoi
   * GetHeader
   atof
   fgets
   fclose
2. Function name: *GetHeader(char *Filename)*
   Calls: fopen
   malloc
   fgets
   free
   fclose
3. Function name: *main(int argv, char *argv[])*
   Calls:
   * InitSI
   * ReadRutCorr
   * DisplayHelp
   * MonitorMode
   _clearscreen
   printf
   free
4. Function name: *ProcessFunctions(unsigned char scan)*
   Calls:
   *Upload
   *FlushPort
   fopen
   *serrecv
   fputc
   *EscapeHit
   fclose
   *sersend
   *serclose_q

*SERCLOSE
*SetCursor
*CommParam
*OpenComm
*DisplayCommParam
* ScrollWin
printf
gets
sscanf
*KeyPadMode
*LoadedCorr
*TestMode
*ProfileMode2
*ProfileMode3
*SDProfile1
*Mode6
*IRI_RUTMode
*DisplayHelp

5. Function Name: *DisplayHelp*
   Calls:
   printf
   *DisplayCommParam
6. Function Name: *DistanceMode*
   Calls:
   *SendCommand
   *GetString
   sscanf
   *SetCursor
   printf
7. Function name: *GetInternalSPF*
   Calls:
   *SendCommand
   *GetString
   sscanf
8. Function name: *GetSCKPMI*
   Calls:
   *SendCommand
   *GetString
   sscanf
9. Function name: *SetInternalSPF*
   Calls:
   *SendCommand
   *Delay
   sprintf
10. Function name: *SetAcUnit*
    Calls:
    sprintf
    *SendCommand
    *GetString
    *sersend
    *FlushPort

11. Function name: *SetSdMode*
    Calls: *SendCommand
        *GetString
        sprintf
        *sersend
        *FlushPort
12. Function name: *SetExtractRatio*
    Calls: sprintf
        *SendCommand

13. Function name: *SetParam*
    Calls: *SetInternlSPF
        *SetExtractRatio
14. Function name: *GetAcoustic*
    Calls: *ClearRows
        *SetCursor
        printf
        gets
        strcpy
        *SelectAcoustic
15. Function name: *SelectAcoustic*
    Calls: None
16. Function name: *GetFileName*
    Calls: *time
        *ClearRows
        *SetCursor
        printf
        gets
        getch
        sprintf
        fclose
        fopen
        strcpy
        fprintf
17. Function name: *GetSPF*
    Calls: *ClearRows
        *SetCursor
        printf
        gets
        fprintf
18. Function name: *GetDistance*
    Calls: *SetCursor
        *ClearRows
        printf
        gets
        sscanf

19. Function name: *DisplayPoints*
    Calls: *ClearRows
        *SetCursor
        printf
20. Function name: *ProfileMode2*
    Calls: *SetFilter
        *ClearRows
        *Border
        *SetCursor
        printf
        *DistanceMode
        *SetSdMode
        *GetFileName
        *GetAcoustic
        *SetAcUnit
        *GetDistance
        *StopCollect
        fclose
        *EndMessage
        gets
        sscanf
        *DisplayPoints
        *DisplayStatusForm
        *SetParam
        *SetExtractRatio
        *SendCommand
        *time
        *WaitForFirst
        *CheckTime
        strcpy
        *MasterOutput
        *serrecv
        fprintf
        kbhit
        *CheckComm
        *SaveOrNot
21. Function name: *MasterOutput*
    Calls: fprintf
22. Function name: *Ckecktime*
    Calls: clock
23. Function name: *KeyPadMode*
    Calls: *SendCommand
        *FlushPort
        _ClearScreen
        _setvideomode
        *SetCursor
        *Border
        kbhit
        *readkey
        *sersend
        printf

*serrecv
putch
_setvideomode

24. Function name: *TestMode*
Calls: *ClearRows
*Border
*SetCursor
printf
*SetSdMode
*SendCommand
*Flushport
*SelectAcoustic
*SetAcUnit
*SetParam
*SendCommand
*WaitForFirst
*serrecv
*sersend
*readkey
*Delay

25. Function name: *SetFilter*
Calls: *SendCommand
sprintf
printf

26. Function name: *ReadRutCorr*
Calls: fopen
fscanf
fclose

27. Function name: *SaveRutCorr*
Calls: fopen
fprintf
fclose

28. Function name: *LoadedCorr*
Calls: *ReadRutCorr
*DispCorr
printf
*readkey
*TestMode
*SaveRutCorr
*DispCorr
*SetCursor
*sersend

29. Function name: *LoadedCorr*
Calls: ReadRutCorr
DispCorr
TestMode
readkey
SaveRutCorr
DispCorr
SetCursor
sersend

29. Function name: *DispCorr*
Calls: printf

Warnier orr Diagram

```
1    2    3    4    5    6    7    8    9    10
main
        InitSi
                GetHeader |
        MonitorMode
                SetCursor |
                OpenComm
                        serinit_q |
                        serinit   |
                ProcessFunctions
                        Upload
                                EscapeHit
                                        kbhit |
                                sersend  |
                                serrecv  |
                                FlushPort
                                        sersend |
                        FlushPort
                                sersend |
                        serrecv |
                        EscapeHit
                        sersend |
                        serclose_q |
                        SetCursor  |
                        CommParam |
                        OpenComm
                                serinti_q  |
                                serinit    |
                        DisplayCommParam |
                        ScrollWin |
                        KeyPadMode
                                SendCommand
                                        FlushPort
                                                sersend |
                                        EscapeHit
                                                kbhit |
                                        WaitChar
                                                serrecv |
                                FlushPort
                                        sersend |
                                SetCursor |
                                Border
                                        SetCursor |
                                        WriteCh   |
                                readkey |
                                sersend |
                                serrecv |
                        LoadedCorr
                                ReadRutCorr |
                                DispCorr   |
                                TestMode
                                        ClearRows |
                                        Border
```

```
        SetCursor |
        WriteCh   |
SetCursor |
SetSdMode
        SendCommand
                FlushPort
                        serrecv |
                EscapeHit
                        kbhit |
                WaitChar
                        sersend |
        GetString
                WaitChar
                        sersend |
                EscapeHit
                        kbhit |
                FlushPort
                        serrecv |
        sersend |
        FlushPort
                serrecv |
SendCommand
        FlushPort
                serrecv |
        EscapeHit
                kbhit |
        WaitChar
                sersend |
FlushPort
        serrecv |
SelectAcoustic |
SetAcUnit
        SendCommand
                FlushPort
                        serrecv
                EscapeHit
                        kbhit |
                WaitChar
                        sersend |
        GetString
                WaitChar
                        sersend |
                EscapeHit
                        kbhit |
                FlushPort
                        serrecv |
        sersend |
        FlushPort
                serrecv |
SetParam
        SetInternalSPF
                SendCommand
                        FlushPort
                                serrecv |
                        EscapeHit
```

```
                                        kbhit |
                                    WaitChar
                                        sersend |
                                delay |
                            SetExtractRatio
                                SendCommand
                                    FlushPort
                                        serrecv |
                                    EscapeHit
                                        kbhit |
                                    WaitChar
                                        sersend |

                    WaitForFirst
                        readkey |
                        SetCursor |
                        serrecv  |
                        ClearRows |
                    serrecv |
                    sersend |
                    readkey |
                    delay  |

            readkey   |
            SaveRutCorr |
            SetCursor |
            sersend  |
TestMode
            ClearRows |
            Border
                    SetCursor |
                    WriteCh  |
            SetCursor |
            SetSdMode
                    SendCommand
                        FlushPort
                            serrecv |
                        EscapeHit
                            kbhit |
                        WaitChar
                            sersend |
                    GetString
                        WaitChar
                            sersend |
                        EscapeHit
                            kbhit |
                        FlushPort
                            serrecv |
                    sersend  |
                    FlushPort
                        serrecv |
                SendCommand
                    FlushPort
                        serrecv |
                    EscapeHit
```

```
                    kbhit |
            WaitChar
                    sersend |
    FlushPort
            serrecv |
    SelectAcoustic |
    SetAcUnit
            SendCommand
                    FlushPort
                            serrecv |
                    EscapeHit
                            kbhit |
                    WaitChar
                            sersend |
            GetString
                    WaitChar
                            sersend |
                    EscapeHit
                            kbhit |
                    FlushPort
                            serrecv |
            sersend |
            FlushPort
                    serrecv |
    SetParam
            SetInternalSPF
                    SendCommand
                            FlushPort
                                    serrecv |
                            EscapeHit
                                    kbhit |
                            WaitChar
                                    sersend |
                    delay |
            SetExtractRatio
                    SendCommand
                            FlushPort
                                    serrecv |
                            EscapeHit
                                    kbhit |
                            WaitChar
                                    sersend |
    WaitForFirst
            readkey |
            SetCursor |
            serrecv  |
            ClearRows |
    serrecv |
    sersend |
ProfileMode2
        SetFilter
            SendCommand
                    FlushPort
                            serrecv |
```

```
                    EscapeHit
                        kbhit |
                    WaitChar
                        sersend |
ClearRows |
Border
        SetCursor |
        WriteCh |
SetCursor |
DistanceMode
        SendCommand
                FlushPort
                        serrecv |
                EscapeHit
                        kbhit |
                WaitChar
                        sersend |
        GetString
                WaitChar
                        sersend |
                EscapeHit
                        kbhit |
                FlushPort
                        serrecv |
        SetCursor |
SetSdMode
        SendCommand
                FlushPort
                        serrecv |
                EscapeHit
                        kbhit |
                WaitChar
                        sersend |

        GetString
                WaitChar
                        sersend |
                EscapeHit
                        kbhit |
                FlushPort
                        serrecv |
        sersend |
        FlushPort
                serrecv |
GetFileName
        ClearRows |
        SetCursor |
GetAcoustic
        ClearRows |
        SetCursor |
        SelectAcoustic |
SetAcUnit
        SendCommand
                FlushPort
                        serrecv |
```

```
                    EscapeHit
                            kbhit |
                    WaitChar
                            sersend |
            GetString
                    WaitChar
                            sersend |
                            EscapeHit
                                    kbhit |
                            FlushPort
                                    serrecv |
            sersend |
            FlushPort
                    serrecv |
GetDistance
            SetCursor |
            ClearRows |
StopCollect
            sersend |
            FlushPort
                    serrecv |
EndMessage
            sersend  |
            SetCursor |
            Border
                    SetCursor |
                    WriteCh   |

            ClearRows |
DisplayPoints
            ClearRows |
            SetCursor |
DisplayStatusForm
            ClearRows |
            SetCursor |
SetParam
            SetInternalSPF
                    SendCommand
                            FlushPort
                                    serrecv |
                            EscapeHit
                                    kbhit |
                            WaitChar
                                    sersend |
                    delay |
            SetExtractRatio
                    SendCommand
                            FlushPort
                                    serrecv |
                            EscapeHit
                                    kbhit |
                            WaitChar
                                    sersend |
SetExtractRatio
```

```
                SendCommand
                        FlushPort
                                serrecv |
                        EscapeHit
                                kbhit |
                        WaitChar
                                sersend |

        SendCommand
                FlushPort
                        serrecv |
                EscapeHit
                        kbhit |
                WaitChar
                        sersend |
        WaitForFirst
                readkey |
                SetCursor |
                serrecv  |
                ClearRows |
        Checktime |
        MasterOutput |
        serrecv |
        CheckComm
                SerialPortStatus |
                SetCursor |
        SaveOrNot
                ClearRows |
                SetCursor |
ProfileMode3
        GetMem
                SendCommand
                        FlushPort
                                serrecv
                        EscapeHit
                                kbhit |
                        WaitChar
                                sersend |
                GetString
                        WaitChar
                                sersend |
                        EscapeHit
                                kbhit |
                        FlushPort
                                serrecv
        ClearRows |
        Border
                SetCursor |
                WriteCh  |
        SetCursor |
        GetFileName
                ClearRows |
                SetCursor |
        GetAcoustic
```

```
        ClearRows |
        SetCursor |
        SelectAcoustic |
GetSPF
        ClearRows |
        SetCursor |
GetDistance
        SetCursor |
        ClearRows |
DisplayPoints
        ClearRows |
        SetCursor |
DisplayStatusForm
        ClearRows |
        SetCursor |
SetExtractRatio
        SendCommand
                FlushPort
                        serrecv |
                EscapeHit
                        kbhit |
                WaitChar
                        sersend |
SetAcUnit
        SendCommand
                FlushPort
                        serrecv |
                EscapeHit
                        kbhit |
                WaitChar
                        sersend |
        GetString
                WaitChar
                        sersend |
                EscapeHit
                        kbhit |
                FlushPort
                        serrecv |
        sersend |
        FlushPort
                serrecv
InitRut |
SetInternalSPF
        SendCommand
                FlushPort
                        serrecv |
                EscapeHit
                        kbhit |
                WaitChar
                        sersend |
        delay |
```

```
SendCommand
        FlushPort
                serrecv |
        EscapeHit
                kbhit |
        WaitChar
                sersend |
sersend |
FlushPort
        serrecv |
readkey |
WaitForFirst
        readkey |
        SetCursor |
        serrecv  |
        ClearRows |

ScrollWin |
DistManage
        sersend |
serrecv  |
WriteSI |
GetRut
        COUNT_TO_INCH |
PostRut
        SendString
                sersend |
        ScrollWin |
        SetCursor |
        PutRoadInfo
                Border
                        SetCursor |
                        WriteCh   |
                SetCursor |
PutRoadInfo
        Border
                SetCursor |
                WriteCh   |
        SetCursor |
PauseRut
        SetCursor |
        Border |
CheckComm
        SerialPortStatus |
        SetCursor |
.StopCollect
        sersend |
        FlushPort
                serrecv |
SaveOrNot
        ClearRows |
        SetCursor |
```

```
        CloseRut |
        EndMessage
                sersend  |
                SetCursor |
                Border
                        SetCursor |
                        WriteCh   |

                ClearRows |
SDProfile1
        ClearRows |
        Border
                SetCursor |
                WriteCh   |
        SetCursor  |
        SetSdMode
                SendCommand
                        FlushPort
                                serrecv |
                        EscapeHit
                                kbhit |
                        WaitChar
                                sersend |
                GetString
                        WaitChar
                                sersend |
                        EscapeHit
                                kbhit |
                        FlushPort
                                serrecv |
                sersend   |
                FlushPort
                        serrecv |
        SetMem
                SendCommand
                        FlushPort
                                serrecv |
                        EscapeHit
                                kbhit |
                        WaitChar
                                sersend |
                GetString
                        WaitChar
                                sersend |
                        EscapeHit
                                kbhit |
                FlushPort
                        serrecv |
        GetFileName
                ClearRows |
                SetCursor |
```

```
GetAcoustic
        ClearRows |
        SetCursor |
        SelectAcoustic |
    SetAcUnit
        SendCommand
            FlushPort
                serrecv |
            EscapeHit
                kbhit |
            WaitChar
                sersend |
        GetString
            WaitChar
                sersend |
            EscapeHit
                kbhit |

            FlushPort
                serrecv |
        sersend |
        FlushPort
            serrecv |
    GetSPF
        ClearRows |
        SetCursor |
    StopCollect
        sersend |
        FlushPort
            serrecv |
    DisplayPoints
        ClearRows |
        SetCursor |
    DisplayStatusForm
        ClearRows |
        SetCursor |
    SetParam
        SetInternalSPF
            SendCommand
                FlushPort
                    serrecv |
                EscapeHit
                    kbhit |
                WaitChar
                    sersend |
            delay |
        SetExtractRatio
            SendCommand
                FlushPort
                    serrecv |
                EscapeHit
                    kbhit |
                WaitChar
```

```
                                                    sersend |
            SendCommand
                    FlushPort
                            serrecv |
                    EscapeHit
                            kbhit |
                    WaitChar
                            sersend |
            WaitForFirst
                    readkey |
                    SetCursor |
                    serrecv  |
                    ClearRows |
            serrecv |
            sersend
            CheckComm
                    SerialPortStatus |
                    SetCursor |
            SaveOrNot
                    ClearRows |
                    SetCursor |

            EndMessage
                    sersend  |
                    SetCursor |
                    Border
                            SetCursor |
                            WriteCh   |

                    ClearRows |

Mode6
            ClearRows |
            Border
                    SetCursor |
                    WriteCh   |
            SetCursor |
            DistanceMode
                    SendCommand
                            FlushPort
                                    serrecv |
                            EscapeHit
                                    kbhit |
                            WaitChar
                                    sersend |
                    GetString
                            WaitChar
                                    sersend |
                            EscapeHit
                                    kbhit |
                            FlushPort
```

```
                    serrecv |
        SetCursor |
SetSdMode
        SendCommand
                FlushPort
                    serrecv |
                EscapeHit
                    kbhit |
                WaitChar
                    sersend |
        GetString
                WaitChar
                    sersend |
                EscapeHit
                    kbhit |
                FlushPort
                    serrecv |
        sersend|
GetFileName
        ClearRows |
        SetCursor |
SetAcUnit
        SendCommand
                FlushPort
                    sersend |
                EscapeHit

                    kbhit |
                WaitChar
                    sersend |
        GetString
                WaitChar
                    sersend |
                EscapeHit
                    kbhit |
                    sersend |
        sersend |
        FlushPort
                sersend |
GetSPF
        ClearRows |
        SetCursor |
GetDistance
        SetCursor |
        ClearRows |
StopCollect
        sersend |
        FlushPort
                sersend |
EndMessage
        sersend |
        SetCursor |
```

```
                Border
                        SetCursor |
                        WriteCh   |
                ClearRows |
        DisplayPoints
                ClearRows |
                SetCursor |
        DisplayStatusForm
                ClearRows |
                SetCursor |
        SetParam
                SetInternalSPF
                        SendCommand
                                FlushPort
                                        sersend |
                                EscapeHit
                                        kbhit |
                                WaitChar
                                        sersend |
                        delay |
                SetExtractRatio
                        SendCommand
                                FlushPort
                                        sersend |
                                EscapeHit
                                        kbhit |
                                WaitChar
                                        sersend |
        SendCommand
                FlushPort
                        sersend |

                EscapeHit
                        kbhit |
                WaitChar
                        sersend |
        WaitForFirst
                readkey |
                SetCursor |
                serrecv  |
                ClearRows |
        serrecv   |
        MasterOutput |
        CheckComm
                SerialPortStatus |
                SetCursor |
        SaveOrNot
                ClearRows |
                SetCursor |
IRI_RUTMode
        ClearRows |
```

```
Border
        SetCursor |
        WriteCh  |
SetCursor  |
GetFileName
        ClearRows |
        SetCursor |
GetAcoustic
        ClearRows |
        SetCursor |
        SelectAcoustic |
SetAcUnit
        SendCommand
                FlushPort
                        sersend   |
                EscapeHit
                        kbhit |
                WaitChar
                        sersend |
        GetString
                WaitChar
                        sersend |
                EscapeHit
                        kbhit |
                FlushPort
                        sersend |
        sersend |
        FlushPort
                sersend |
SendCommand
        FlushPort
                sersend |
        EscapeHit
                kbhit |
        WaitChar
                sersend |
SetMem
        SendCommand
                FlushPort
                        sersend |
                EscapeHit
                        kbhit |
                WaitChar
                        sersend |
        GetString
                WaitChar
                        sersend |
                EscapeHit
                        kbhit |
                FlushPort
                        sersend |
        FlushPort
                sersend |
```

```
GetSPF
        ClearRows |
        SetCursor |
GetDistance
        SetCursor |
        ClearRows |
StopCollect
        sersend |
        FlushPort
                sersend |
EndMessage
        sersend  |
        SetCursor |
        Border
                SetCursor |
                WriteCh   |
        ClearRows |
DisplayPoints
        ClearRows |
        SetCursor |
DisplayStatusForm
        ClearRows |
        SetCursor |
SetParam
        SetInternalSPF
                SendCommand
                        FlushPort
                                sersend |
                        EscapeHit
                                kbhit |
                        WaitChar
                                sersend |
                delay |
        SetExtractRatio
                SendCommand
                        FlushPort
                                sersend |

                        EscapeHit
                                kbhit |
                        WaitChar
                                sersend |
WaitForFirst
        readkey |
        SetCursor |
        serrecv  |
        ClearRows |
GetFloat
        serrecv |
ScrollWin |
GetMsg
        serrecv |
```

```
                    sersend |
                    serrecv |
                    CheckComm
                            SerialPortStatus |
                            SetCursor |
                    SaveOrNot
                            ClearRows |
                            SetCursor |
            DisplayHelp
                    DisplayCommParam |
    ScrollWin  |
    readkey |
    sersend |
    serclose |
    serclose_q |
    serrecv |
ReadRutCorr |
DisplayHelp
    DisplayCommParam |
```

Notes on Variables

- typedef **SERIAL STATUS**

```
typedef union {
    unsigned int w;
    struct {
      /* modem */
      unsigned int RLSD:1;
      unsigned int RI:1;
      unsigned int DSR:1;
      unsigned int CTS:1;
      unsigned int DRLSD:1;
      unsigned int TERD:1;
      unsigned int DDSR:1;
      unsigned int DCTS:1;

      /* line */
      unsigned int DataReady          :1;
      unsigned int OverrunError       :1;
      unsigned int ParityError        :1;
      unsigned int FrameError         :1;
      unsigned int BreakDetect        :1;
      unsigned int THREmpty                 :1;
      unsigned int TSREmpty           :1;
      unsigned int TimeOut            :1;
      } b;
    } SERIALSTATUS;
```

The above typedef is used when you need to use int14 function 03. This function is used to read the serial port status. Refer to the Function SerialPortStatus. A 16 bit value is returned when a call is made to SerialPortStatus. The above data structure SERIALSTATUS is returned.

The status of the port which is a unsigned integer. When this interrupt is called a value is returned an register ax. The upper byte register **ah** contains.

Bit 0: Data Ready
Bit 1: OverrunError
Bit 2: Parity Error
Bit 3: Framing Error
Bit 4: Break discovered
Bit 5: Transmission hold register empty
Bit 6: Transmission shift register empty

The lower byte register **al** contains

Bit 0: Modem ready to send status change
Bit1: Modem on status change
Bit2: . Telephone ringing status change
Bit3: Connection to receiver status change
Bit4: Modem ready to send
Bit5: Modem on
Bit6: Telephone ringing
Bit7: Connection to receiver modem

- int **iFrame**

This variable indicates that there has been some Frame error. iFrame is incremented whenever there is a Frame error.

- int **iOverrun**

This variable indicates that there has been some Overrun error. iOverrun is incremented whenever there is an Overrun error.

- File ***fRut**

It is a file descriptor which is used to write the Rut.

- File ***fSI**

It is a file descriptor which is used to write the serviceability index

- File ***fWSV;**

It is a file descriptor which is used to write the Walker slope variable.

- **iRutCnt**

This is the Rut count....not very sure about this variable. It is used and modified in PostRut.

- **iRutMethod**

This is a value of the Rutmethod which is either

* 0 Do not do real time rut
* 1 South Dakota
* 2 String-Line Right
* 4 String-Line Left
* 6 String-Line Right & Left
* 8 Right Rut
* 12 New Average (String-Line Left & Right Rut)
* 28 Method 12; output to serial ports; does not write to disk.

- **bExtraComm**

bExtraComm is the value of iRutMethod &0x10
bExtraComm is true only if the iRutMethod is 28....that is Method 12 mentioned above

- **SIDist**

SIDist is the value of the distance in feet when every value of SI comes in from the SIOMETER
SIDist = GetMem(0x10529eL)/3; //either 1056 or 528; This is given in mode3.c
After every SIDist, an SI signal comes in.

- **nRutsPerDMI**

The nRutsPerDMI is 3. A Rut signal , that is acoustic data comes in every 4 feet. A distance signal is sent every 12 feet. Therefore 3*14 = 12. After 3 Rut signals a distance signal is sent. Therefore the variable nRutsPerDMi is the value 3.

- **PortNo**

This variable is defined in file talk.h. This is used to define the port number. This is the port number that is used for communication. For sending or receiving(reading) we need a port to so.

- **Databits**

This specifies the number of databits required for forming the frame
It stores the number of databits that are being used for transmission. It is typically either 7 or 8.

- **Stopbits**

This specifies the number of stopbits required for forming the frame

Stopbits It is the number of stop bits that are being used. It is typically 1 or 2.

- **SupportedBaudRate**

This is an array which gives all the supported baud rates

- **DisplayMode**

This variable is used to define whether the program is in ASCIIMODE or DISPLAYMODE

- **DataCollectMode**

It is a variable that indicates what types of rut measuring methods is bring used. Different types of rut measuring methods are South Dakota Method, Construction Mode etc.

- char ***szMode[NMODES]**

This is just a character array used for printing. It has a character string used for printing for all rhw 7 modes.

```
szMode[ACC_ACOUST] = "Acceleration and acoustic data";
szMode[ACCONLY] = "Acceleration only";
szMode[SI_AVGSPD] = "SI values, acoustic data, and average speeds";
szMode[CONSTR] = "Construction";
szMode[SDK] = "South Dakota";
szMode[TWOCH] = "Acceleration and Laser"
szMode[IRI_RUT] = "IRI and RUT"; //[70]
```

- **szActiveAcoustic**

This variable is an array that is used to store the acoustic devices that are active i.e. are being used. The acoustic devices are mounted on the rut bar that is attached to the vehicle that is being used for this purpose. These acoustic devices are numbered 1 through 5. So the variable szActiveAcoustic is used to store the number that corresponds to the acoustic device that is being used for the data collection purpose.

- char **str[80]**

This is used as a temporary character buffer. Whenever we need to send something to the siometer we first copy it into this buffer and then send it to the siometer.

- unsigned char **AcousticMask**

This is a variable that stores the numbers of the acoustic devices that have been selected to collect the data. If the value in binary is 00011111 then the acoustic devices 1, 2, 3, 4 and 5 have been selected for the collection of the data. The acoustic devices are numbered 1 through 5 from left to right. A laser can also be used and be numbered as 6.

- **bCountAcoustic**

If bCountAcoustic is true then count acoustic instead of acceleration.

- **bDistMode**

bDistMode gets the value of IsDistance. IsDistance = TRUE indicates that it is distance mode. IsDistance = FALSE indicates that it is timemode.
Example calls.
bDistMode = DistanceMode(TRUE);
bDistMode = DistanceMode(FALSE);

- ***fData**

It is a pointer to a file

- **bAbort**

bAbort is used to know if any process is aborted. If any process like sending a file is aborted then it is set to TRUE, else it is set to false. A way of setting bAbort is by pressing the escape key on the keyboard.

- **DatFile**

This is a variable that is used to store the name of a file.

- **BaudRate**

  It stores the baud rate that has been selected for the communication. Baudrate is normally specified in number of bits per second.

- **Parity**

  This variable is used to store the type of parity that is being used for communication. It can be either ODD parity or EVEN parity or No parity. Basically parity bits are check bits that are used to see if the received data is correct or not.

- **bPause**

This variable is used to print " Pause" on the standard input.

- **bMetric**

  This variable is to know whether the metric that we are using to measure the distance is Kilometers or feet. If this variable is set to 1(TRUE) then distance is printed in Kilometers else if it is false then it is reset to zero(FALSE) then the distance is printed in feet.

- **szDefAc**

This variable is used to store the numbers of acoustic devices that are being used.

- **RutCorr.Loading**

This array is used to store the differences in acoustic readings for the loaded and unloaded conditions. An unloaded condition is one in which all the persons in the vehicle get out of the vehicle and the acoustic readings are collected. A loaded condition is one in which all the members get in to the vehicle and the acoustic readings are collected.

- **sRutCorr**

  This variable is used to store the name of the file called "RUTCORR.INI". The name stands for RUT CORRECTION INITIALISATION.

- **nInline**

This is the number of the values in a line. When the number of values in a line becomes 16 then it is set to 0. This is because the format of a file is so described that it contains 16 values in a line. So after reading 16 value we go to the next line and set nInline to 0.

- **TWOCH**

  "Acceleration and Laser"

- **CONSTR**

  "Construction" This is one of the modes for the collection of the data. Others include SoutDakota Mode, Acceleration and Laser mode etc..

- **SI_AVGSPD**

  "SI values, acoustic data, and average speeds";

- **bSysGenFileName**

This is a variable that tells whether the name of file to store the data collected is entered by the user. If the user does not enter the name of the file then the program generates a filename which constitutes of system time in month, day, hours, minutes and seconds.

- **bComment**

This variable if set asks for comment to be entered. Depending on the variable header it asks the user to enter one or two lines of comment. If header is set to 1 then it asks the user to enter two lines of comment else if the header is reset to 0 then it asks the user to enter just 1 line of comment. If comment is set to 'N' then the user is not asked to enter any comment.

- **Header**

It gets the value of the header from a file. The header is stored in the beginning of the file.

- **RangeLimits**

This is an array storing the maximum value of each range of value that can be received.

- **iRutSecLen**

This specifies the rut report interval in feet. The options are 528 feet and 1056 feet.

- **Param**

This is a variable used to store the tokens read from the file whose name is stored in the variable called 'fn'. The tokens are that are stored in this variable are predefined tokens.

- **value**

This variable stores the value to which is used to set another variable.

- **RutCorr.Mounting**

This contains the mounting offsets of the five acoustic devices in inches. We need this because the bar on which the acoustic devices are mounted may not be horizontal and the devices may have different offsets.

- **DisplayMode**

This variable is used to define whether the program is in ASCIIMODE or DISPLAYMODE

- **bDistMode**

bDistMode gets the value of IsDistance. IsDistance = TRUE indicates that it is distance mode. IsDistance = FALSE indicates that it is timemode.
Example calls.
bDistMode = DistanceMode(TRUE);
bDistMode = DistanceMode(FALSE);

- **bAbort**

bAbort is used to know if any process is aborted. If any process like sending a file is aborted then it is set to TRUE, else it is set to false. A way of setting bAbort is by pressing the escape key on the keyboard.

- **SDK**

SouthDakotaMode

- **F5**

If this is set to 'N' then mode changing is disabled

- **F2**

If this is set to 'N' then changing communication parameters is disabled.

- **RUT_SEC_LEN**

This represents the rut section length over which the rut is reported. If this is set to say, 528 feet then rut report interval is 528 feet.

- **COM3IRQ**

This is used to store an integer. If this is set to 2 then it means that use IRQ2 for COM3.

Scancodes

Some Info on Scan codes.

When the user presses any key on the keyboard, an electrical impulse, which identifies the location of the key is generated. This signal is handled by the Keyboard processor, which is located inside the Keyboad itself. Generally this processor is an Intel 8084 chip. If you are using an AT class of computer then the communication is handled by an Intel 8042 chip. This allows bi-directional communication between CPU and Keyboard. Earlier PC's and XT's do not have this capability.

Converting the scancode

The Keyboard processor converts the electrical impulse indicating the position into a number called scancode. This scan code is passed to the computer. The transfer is done serially, since the cable that connects the Keyboard to the computer has only one data line. The communication is synchronous.

Scan code are also generated when the key is released. This is important because the computer needs to know if the key has been released or still pressed. This helps to differentiate the situations like typing the capital letters and also trying to reboot the computer. For rebooting the computer needs to know that all the three keys namely <Ctrl><Alt><Delete> are pressed together.

Each time the keyboard is pressed a hardware interrupt IRQ1 is executed. The keyboard handler receives these scancodes one when a key is pressed and one when key is released and converts these into corresponding ASCII character codes, which can be read by application that is currently running.
Different keyboards use different sets of scancodes. So they by themselves are unusable. So these scan codes are converted into ASCII codes, which are standard on all computers.