DESIGN OF AN EASY-TO-USE, HOST-INDEPENDENT DATA ACQUISITION SYSTEM

by

DURWIN DUANE NIGUS

B.S., Kansas State University, 1987

A THESIS

submitted in partial fulfillment of the

requirements for the degree

MASTER OF SCIENCE

ELECTRICAL AND COMPUTER ENGINEERING

KANSAS STATE UNIVERSITY
Manhattan, Kansas

1989

Approved by:

Stephen A. Dyer

Major Professor

TABLE OF CONTENTS

TABLE OF CONTENTS (cont.)

TABLE OF CONTENTS (cont.)

LIST OF FIGURES

LIST OF FIGURES (cont)

LIST OF FIGURES (cont)

LIST OF TABLES

LIST OF TABLES (cont.)

LIST OF TABLES (cont.)

LIST OF SCHEMATICS

LIST OF SCHEMATICS (cont.)

SCHEMATIC PAGE

LIST OF SCHEMATICS (cont.)

ACKNOWLEDGEMENTS

There are numerous people that I must acknowledge.

First, I wish to acknowledge my major professor, Dr. Stephen A. Dyer, who was responsible for finalizing the specifications for this project and assisting with its design throughout its development.

I also wish to thank Dr. Richard R. Gallagher and Dr. Gale G. Simons for being on my committee.

Two other people in particular deserve acknowledgement: Ken Boyer and Terry Hull. Mr. Boyer's assistance with numerous design aspects and Mr. Hull's suggestions for bus-design and protocol were indispensable.

And acknowledgement is certainly in order for my father, Duane Nigus. My interest in electronics and my decision to pursue electrical engineering as a career stems from the many, many hours we spent discussing electronics when I was young.

And what set of acknowledgements would be complete without mentioning one's spouse? Special thanks are in order to my wife, Pamela, whose patience and understanding during the development of this project made the whole thing possible.

CHAPTER ONE

INTRODUCTION

The EECE department at Kansas State University frequently has a need for signal data acquisition. However, there are few data acquisition systems in the department. Of the systems presently available, their use is limited to one type of computer. This is fine when the data is to be analyzed on the same computer which was used to acquire it. However, when the data is to be analyzed on a computer <u>other</u> than the one used to acquire it, the data must be transferred, which is sometimes a cumbersome task.

One solution is to outfit every computer in the department with its own data acquisition system (DAS). However, the cost of this solution is prohibitive. Another solution is to obtain a DAS that can be used with any computer. Unfortunately, a flexible, high-quality DAS of this nature is not commercially available. Therefore, the purpose of this thesis is to present the design of a DAS that can be used with many different types of computers.

There are many steps involved between the proposal of the DAS and its use. These steps include:

1) System proposal.
2) Hardware-level design of the system.
3) Construction of the prototype.
4) Testing of the prototype circuit.
5) Development of system control software.
6) Use of the system.
7) Design of additional boards to augment the system's capabilities.

This thesis covers steps 1 through 4, and includes suggestions for algorithms used in step 5.

Before giving a description of the steps taken during the design of the system, a brief explanation of the "name" of the system is in order. Throughout this thesis the data acquisition system is referred to as, for lack of a better name, the DAS. Strictly speaking, a data acquisition system and its acronym, DAS, refer to a system whose function is limited to acquiring data. However, this system's specifications (which will be presented shortly) indicate that the system is capable of both acquiring and generating data which is analog or digital in nature. Therefore, one must keep in mind that though the system is called a DAS, the system has many capabilities beyond what the name implies.

The first step of this thesis project was the system proposal. This proposal consisted of a list of desireable commands and features, which included:

* The DAS must be able to be used with several host computers (i.e. be host-independent);
* All commands for the DAS, such as, setting sample rate, selecting triggers, and selecting sample size, must be able to be provided by the host computer.
* The DAS's operating status must be available to the host computer. This status is to include information about the presence of a board, whether the board is waiting for trigger, or whether the board is ready to send its acquired data.
* All communication to and from the DAS must be in 7-bit ASCII, and include some means of error detection.
* The DAS must be expandable in the sense that additional input/output (I/O) boards, such as digital-to-analog (D/A) conversion and parallel I/O, could be used with this system. This expansion must require a minimum of effort.
* The system should be able to support as many as 16 boards simultaneously.

A proposed set of commands for the system are listed in Appendix B.

Once the specifications were established in the first step, the design of the hardware was considered. It was important that the hardware design encompassed the features proposed in the first step, while remaining flexible enough to permit additional features to be added later. Chapter 2 of this thesis presents an overview of the system's hardware-level design, including a discussion concerning the rationale of the design selected.

The third step of the design was the construction of the circuits designed in step 2. This step involved determining the availability of parts, acquiring these parts, and assembling the circuits. Chapters 3 and 4 provide information about the circuits that were constructed--Chapter 3 covers the system controller, and Chapter 4 covers the analog-to-digital conversion board. In addition, instructions covering the use of the system are included as Appendix A. Appendix D includes information about the construction of the prototype circuit.

The fourth step of the development of the DAS required testing the prototype circuit. Routines used for these tests are included as Appendix F.

The remaining steps of the DAS implementation were not within the scope of this thesis. However, Chapter 5 of this thesis contains suggestions for algorithms that must be generated as part of step 5.

In the event that additional capabilities are desired for the system, boards may be constructed to facilitate these needs. A general procedure for board-design is described in Appendix C.

It is very important to note that the DAS described herein has many capabilities. The purpose of this thesis is not to present an exhaustive list of the system's capabilities, but rather to provide sufficient information about the system to permit these capabilities to be utilized.

CHAPTER TWO

THE DEVELOPMENT OF THE DAS

2.1 Introduction

The purpose of this chapter is to describe the simplest elements of the DAS, and to elaborate on the rationale of the design chosen. This chapter is divided into two sections, where the first section describes the relationship of the system with the host computer, and the second section describes the inner-workings of the DAS.

2.2 Host-to-DAS Interface Development

The first important design criterion was that the DAS was physically independent of the host computer e.g. the DAS must be housed in an enclosure other than the host computer. Physical independence also means the DAS is responsible for its own power supply needs. Therefore, the only connection between the host computer and the DAS is a communication link.

The communication link between the host computer and the DAS serves two purposes. First, the host sends messages to the DAS by way of the communication link. These messages might be commands for the DAS, or they might be digital data to be converted to an analog signal. The second purpose of the communication channel is to send data from the DAS to the host computer. Examples of data sent from the DAS include: system status information and data generated during an analog-to-digital conversion sequence. Hence, the communication link must be bidirectional.

The selection of the bidirectional communication link was very important in order to fulfill the host-independence requirement of the DAS. Although there are several excellent bidirectional communication links available, the one selected for this system was the RS-232. While certainly not the fastest communication link, the RS-232 is available on many computers that might serve as host, including most PC compatibles and the EECE department's VAX 11/750.

Another important design goal specified that the size of the host computer's DAS controlling software be as small as possible. This is important when the DAS is transferred from one type of host computer to another type: the smaller the host's DAS controller software demands are, the quicker the DAS can be implemented with a new host computer. The duties of the DAS controlling software is to both compose messages sent to the DAS and to decipher responses received from the DAS. Therefore, the size and complexity of the DAS controlling software is directly related to the complexity of the messages to and from the DAS.

One way to reduce the complexity of the messages between the host and DAS was to delegate the duties of message deciphering to the DAS itself. This reduces the software duties of the host to composing mnemonic commands that the DAS can decipher and implement.

The design of the DAS to this point can be summarized as follows. The DAS is physically independent of the host computer, such that the only connection with the host computer is an RS-232 communication link. In order to reduce the complexity of the host computer's DAS control software, the duty of deciphering and implementing the commands was delegated to the DAS.

2.3 Development of the DAS's Internal Structure

The development of the DAS's internal structure took into account many of the design goals. One of the design goals specified that signal I/O (both digital and analog) be handled by removable boards. This permits boards of varying function and specification to be installed in the

DAS as they are needed. Specifying that the boards be removable permits additional boards to be constructed and used with the DAS as needed. Fig. 2.3.1. illustrates the design of a system consisting of removable boards and a communication link with a host computer.

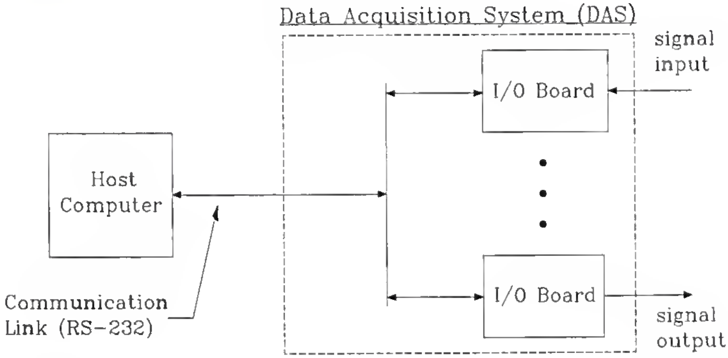


Figure 2.3.1 The block diagram of a DAS consisting of removable I/O boards.

The design depicted in Fig. 2.3.1 requires each I/O board in the DAS to be responsible for many things, including:

communication with the host;
decoding command messages;
controlling board-specific-circuitry;
retaining data until retrieved;
assembling return messages to host;
provide power supply conditioning.

All of these duties could be handled on each board under the direction of a microprocessor. However, this results in a good deal of redundancy between boards e.g. the microprocessor and the RS-232 interface circuit. One way to reduce this redundancy between boards is to delegate many of these duties to a central controller. The central controller, or more specifically the "system front end", would be responsible for the following:

communication with the host;
decoding command messages for each board;
controlling each board;
assembling and returning messages to host.

In other words, the system front-end is responsible for receiving commands from the host computer and sending the appropriate control signals to the I/O boards. The link between the system front-end and each of the boards is called the system bus. The system bus consists of data, address, and other lines necessary to communicate with each of the boards. A block diagram of the DAS including the system front-end and the system bus is shown in Fig. 2.3.2.

The DAS includes one or more I/O boards. These boards are responsible for digital and analog functions, such as A/D conversion, D/A conversion, and parallel digital I/O. The I/O boards are controlled by setting

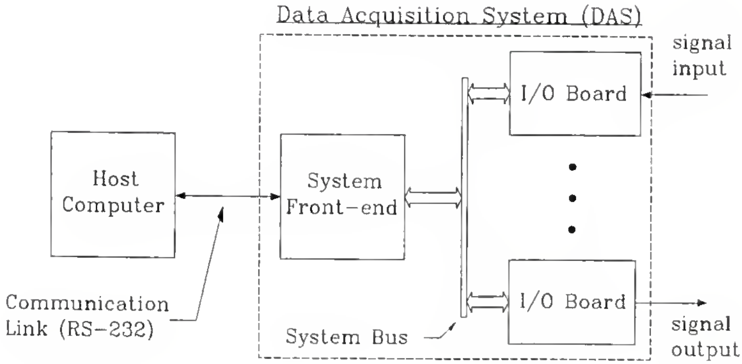


Figure 2.3.2 The DAS including the system front-end and system bus.

registers on the boards, and monitored by reading the board's status registers. Rather than sharing buffer memory among the boards, memory needed for a particular board's operation is located on the board itself.

As shown in Fig. 2.3.3, the I/O boards each consist of three main parts: the bus interface, control and status registers, and the board-specific circuitry.

The control and status registers provide control and monitoring of the board-specific circuitry. For example, the status register on an analog-to-digital (A/D) board includes status bits corresponding to "trigger received"

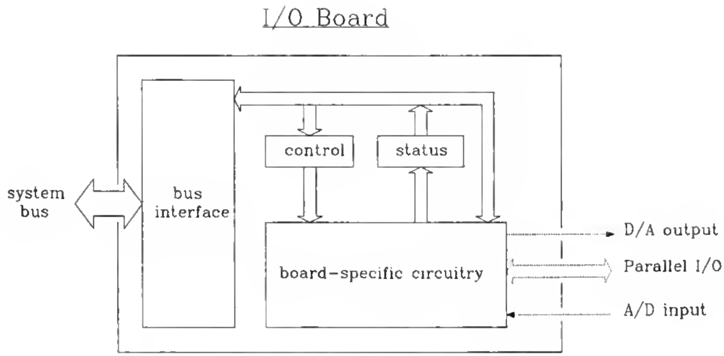


Figure 2.3.3 Block representation of the I/O board.

and "enable acquisition." The control register on an A/D board has control bits that include "enable trigger" and "select bus clock."

The board-specific circuitry depends upon the function of the board. For example, the A/D board has the necessary circuitry associated with analog-to-digital conversion and digital circuitry needed to control it.

All I/O boards used with this DAS must have a system bus interface. This bus interface serves several functions. First, it acts as a buffer between the bus and the digital circuitry on the board. Second, it provides board address decoding. The third function of the bus interface is to provide power supply regulation for the analog circuitry and to provide over-voltage protection for the digital power. A block diagram of the bus interface is shown in Fig. 2.3.4.

Since the boards are controlled by the system front-end, each board's command set must be resident in the system front-end while the board is in use. It is unreasonable to require the system front-end control program to be rewritten every time a new board is constructed. Therefore, a board's command set is stored on that board in an EPROM. The contents of this EPROM must be copied into the system front-end's memory prior to the using the board. When copied, the board's command set is joined with the command sets from the other boards installed in the DAS, and becomes a part of the system's command repertoire.

Though the "EPROM-copy" routine may seem awkward, this approach has many advantages. First, all commands native to a board are stored in that board's EPROM. If additional commands are to be added to a board, then the board's EPROM could be removed and reprogrammed to include the new command. In a similar manner, existing commands could be modified. Another advantage to the "copy" approach is that existing programs would not need to be

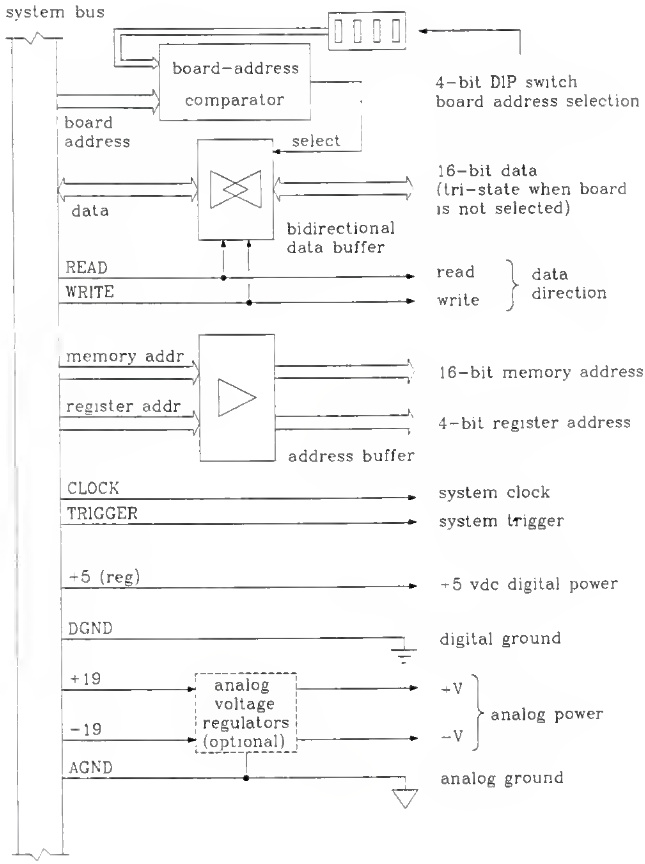


Figure 2.3.4 A block representation of the I/O-board bus interface.

modified when a new board was added to the DAS. The new board's command set would be placed on an EPROM and installed on that board. The new board would then be ready to be used in the DAS.

2.4 Communication with the System

Commands issued by the host are ASCII characters. Each command is composed of two characters (A-Z) followed by any necessary control parameters. A semi-colon (;) is used as the command terminator. A check sum character is also appended to the command instruction to identify the occurrence of an error during transmission. After the host sends a command to the DAS, the host must wait for an acknowledgement string (terminated by a semicolon and a check sum character) from the DAS. If the command sent to the DAS was a request for I/O-board status, the acknowledgement string will consist of information pertaining to the I/O board's status. In the same sense, if the command sent to the is a configuration command, a "command receipt" acknowledgement will be returned.

Messages received from the host are stored in a system front-end memory buffer. After favorable comparison with the check sum, the first two characters of the command are compared with the instruction list in the controller's memory. Once a match is found, the routine corresponding to the instruction is executed. After completing the routine, a return-message is assembled and sent back to the host. The return-message might be a simple acknowledgement that the command was received, or it may be composed of information pertaining to the status of an I/O board. This sequence of operations is shown in Fig. 2.4.1.

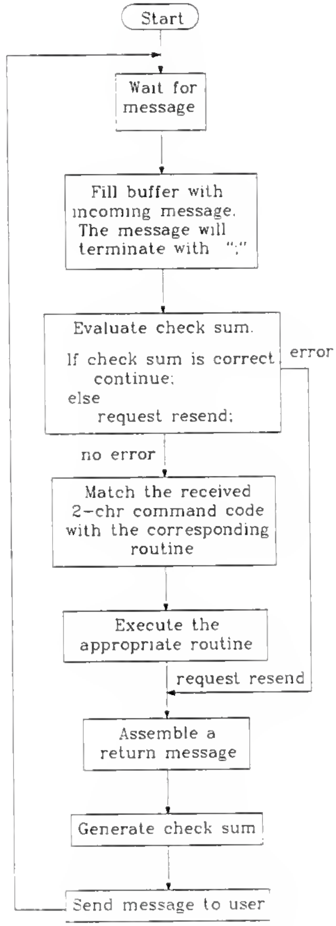


Figure 2.4.1 Operations performed by system controller software.

2.5 Conclusion

A data acquisition system has been described which fulfills many of the design goals that were listed in Chapter 1. The DAS is controlled by command messages that are sent from the host computer over an RS-232 communication link. The mnemonic nature of the DAS's command set reduces the software complexity of the host computer's system control program. The DAS itself is composed of a system front-end and I/O boards, all of which are interconnected by the system bus. The system front-end is responsible for many things, including communications with the host computer and I/O board control. The command set pertaining to a particular I/O board is stored on that board in an EPROM. Before the board can be used, this command set is copied into the system front-end's memory. This command set is then accessed when a command for that board is received by the system.

CHAPTER THREE

THE SYSTEM FRONT-END AND SYSTEM BUS

3.1. Introduction

As described in Chapter 2, the data acquisition system consists of a system front-end and removable I/O boards. The purpose of this chapter is to describe the system front-end. The specifications for the system front-end developed for the DAS are presented in Table 3.1.1.

This chapter presents the system front-end in the following manner. First, the system front-end is described from a user's perspective. This section is followed by description of the system bus and a description of the bus drivers. Following this, algorithms are presented for bus control. The last part of this chapter presents the circuits that make up the system front-end.

Table 3.1.1 Specifications for the DAS system front-end.

Communication:	RS-232, 7-bit, software handshake, adjustable baud rate
System power:	+8 vdc, 2A max. +20 vdc, 2A max. -20 vdc, 2A max. (all power sources may be unregulated)
Maximum number of boards supported simultaneously:	16
other features:	Power for 68HC11EVB is available on a terminal strip (+5V, ±12V); BNC connectors for system clock and system trigger; Over-voltage protection for the +5 volt bus power;
Bus logic levels:	TTL
Bus connectors:	36/72 (0.1"), Vector part number R636-1
Bus power:	+5 volts (regulated) ground for +5V +19 volts (unregulated) -19 volts (unregulated) ground for ±19V supply
Features:	The bus clock and bus trigger provide means for synchronized actions between boards.

3.2. The System Front-end: A User's Perspective

From the user's perspective, the system front-end is the interface device between the host computer and the I/O boards. The connection from the host computer to the DAS is by way of a DB-25 connector, as shown in Fig. 3.2.1. The system controller board (68HC11EVB) is connected to

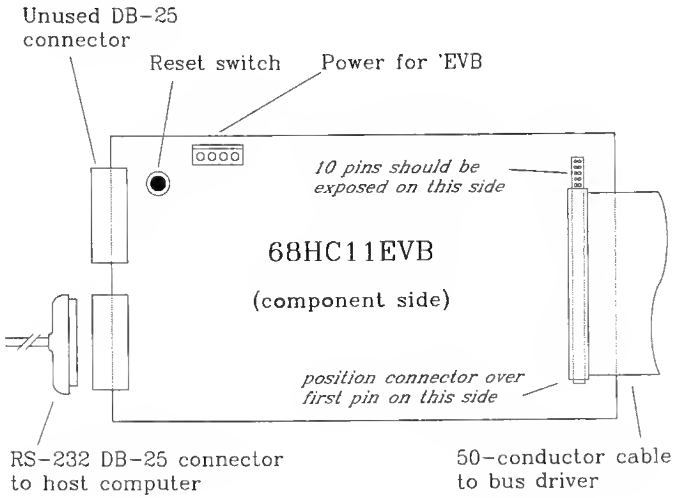


Figure 3.2.1 The connections to the system controller board: the 68HC11EVB.

the bus-driver board by way of a 50-conductor cable, where the proper orientation of this connector on the 68HC11EVB socket is also shown in Fig. 3.2.1. Fig. 3.2.2 shows a top view of the bus-driver board, on which the I/O boards are connected to the system via edge-card connectors. Power for the system is connected by four color-coded banana-style connectors. The power supply connection is made as follows:

```
RED    = +5V
BLACK  =  GND
YELLOW = +20V
GREEN  = -20V
```

Power may be supplied to the system by any power supply (or supplies) capable of the above listed voltages. The amount of current needed for the system depends upon the type and number of boards installed in the system. For most cases (one and two boards installed), the Hewlett-Packard 6236B triple output power supply will suffice. It is very important that the I/O boards are not installed or removed while the system power is on.

Power for the 68HC11EVB is supplied by a compression-type connector on the bus-driver board, where each numbered connector on the bus-driver board is connected to its respectively numbered connector on the 'EVB.

An important thing to verify when using more than one board is that all installed boards have unique addresses. If two boards have the same address, neither of the boards will function properly, and damage to the boards is

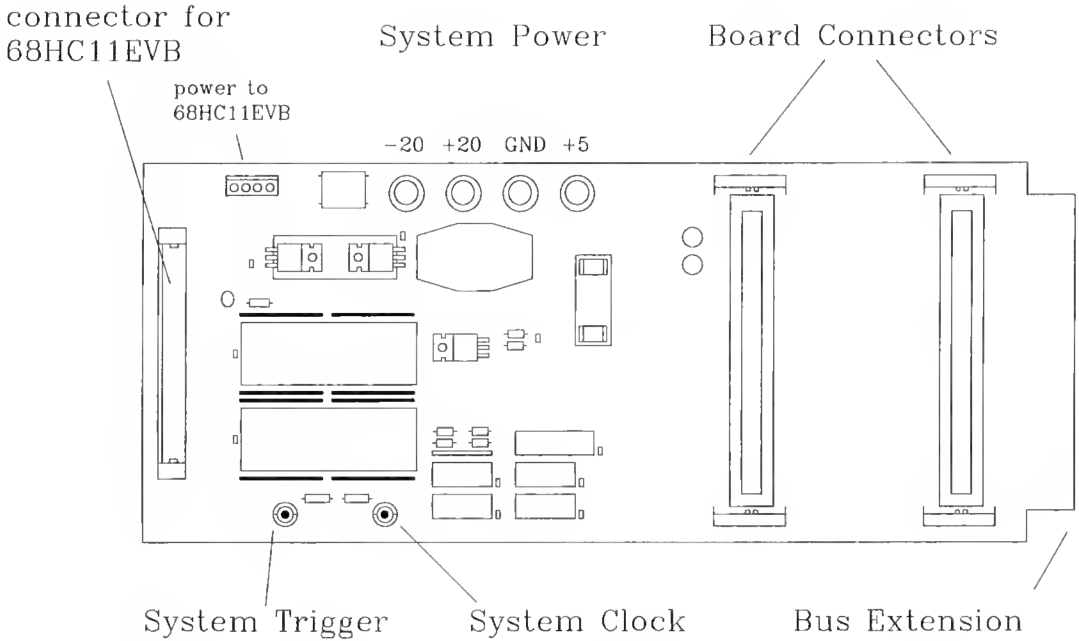


Figure 3.2.2 Top-view of the bus-driver board.

likely.

Two special bus signals are controllable from BNC connections made on the system controller: the system clock and system trigger. The system clock, when enabled, supplies a clock that can be shared by several boards. The system trigger provides a means to trigger several I/O boards simultaneously. Both the system clock and system trigger signals must be TTL compatible.

3.3. The System Bus and Bus Drivers

An important part of the DAS is the system bus. As described in Chapter 2, all communication between the system front-end and the boards installed in the system are made via the system bus. The lines available on the bus are listed in Table 3.3.1.

Table 3.3.1 The lines available on the system bus.

Bus lines	number of bits
board address	4
register address	4
memory address	16
data	16
read/write control	2
bus clock	1
bus trigger	1
power: +5V, +19V, -19V, and numerous grounds	

The 4-bit board address permits up to sixteen boards to be connected to the system simultaneously. The 4-bit register address permits up to sixteen registers on each board to be immediately addressable from the bus.

Sixteen bits of memory address are also provided on the bus. This is useful when addressing on-board memory.

Memory on a board has its own register (addressed by the "register address"), and the "memory address" simply sets the address of the memory accessible from the memory's register.

Data between the system front-end and the boards is carried by the 16-bit data lines. This data is written to or read from the selected board register by appropriate controls from the read/write control lines. The read/write control lines consist of a read-strobe and a write-strobe line. Bus trigger and bus clock both provide a means for synchronized activity between the boards. And finally, power for the boards is available on the system bus. This includes regulated +5V (for digital applications), and unregulated ±19V (for analog applications).

Each of the bus lines are controlled by the system controller via read/write operations to the bus driver ports. A summary of the bus driver ports and their addresses (with respect to the 68HC11EVB) is given in Table 3.3.2. The following is a description for each of these ports.

The 16-bit data lines and 16-bit address lines on the system bus are controlled by ports $A000, $A001, $A005, and $A006. Table 3.3.3 presents the assignments for these

Table 3.3.2 A summary of the bus driver ports and their address with respect to the 68HC11EVB.

Address	Description
$A000 $A001	The "16-bit data" ports.
$A002	The board address and register address port.
$A003	The configuration port for bus driver ports addressed at $A000-$A002.
$A004	Bus control lines, including read, write, and system clock/trigger control
$A005 $A006	The "16-bit address" ports.
$A007	The configuration port for bus driver ports addressed at $A004-$A006.

ports.

The board address and register address lines are controlled by bus driver port $A002. Table 3.3.4 shows how the bits in this port are assigned.

The control lines for the system clock, system trigger, and the read/write strobe lines are accessed via bus driver port $A004. The bit-wise assignment for port $A004 is given in Table 3.3.5. Bits 0 and 1 of bus driver port $A004 are the read and write strobe, where the respective line is made active during a read or write operation to a board register.

Table 3.3.3 The assignments for the address and data bus driver ports.

Bit	Description
D_0-D_7	Port $A000: the 16-bit bus data most significant byte;
D_0-D_7	Port $A001: the 16-bit bus data least significant byte.
D_0-D_7	Port $A005: the 16-bit bus address most significant byte;
D_0-D_7	Port $A006: the 16-bit bus address least significant byte.
	note: D_0 for each port is the least significant bit of that byte.

Bits 4 and 5 of bus driver port $A004 control the bus trigger line. Bit 4 selects the source of the bus trigger: when HIGH, the trigger source is the external trigger connector, and when LOW, the source is bit 5, the system trigger.

Table 3.3.4 The bit-wise assignment of bus driver port $A002.

Bit	Description
D_0-D_3	The 4-bit board address, D_0 is the least significant bit;
D_4-D_7	The 4-bit register address, D_4 is the least significant bit.
	note: D_0 and D_4 are the least significant bits of each 4-bit address.

Table 3.3.5 The bit-wise assignment of bus driver port $A004.

Bit	Description
D_0	bus write line, active LOW
D_1	bus read line, active LOW
D_2-D_3	unused (reserved)
D_4	select external trigger, active HIGH
D_5	system trigger (internal control)
D_6	select external clock, active HIGH
D_7	unused (reserved)

Bit 6 of bus driver port $A004, if HIGH, connects the external clock signal to the bus clock line. When bit 6 is LOW, the bus clock line floats HIGH.

The bus driver devices must be configured as a part of initializing the DAS, where ports $A003 and $A007 are the bus driver configuration ports. The operating mode of the bus driver devices is determined by the value written to these ports, and these values are listed in Table 3.3.6.

Table 3.3.6 Configuration values for bus ports $A003 and $A007.

Configuration description	value
port $A003: bus-data direction = READ form board	$92
port $A003: bus-data direction = WRITE to board	$80
port $A007: all conditions	$80

Note: the bus drivers are inoperative until ports $A003 and $A007 are assigned.

NOTE: When ever the bus data direction is changed (bus port $A003 is written to), the board and register address lines are altered and need to be refreshed.

3.4. Algorithms for System Bus Control

The purpose of this section is to present information that is important to consider when generating algorithms for the system front-end. All algorithmic references are with respect to programs implemented in the system controller. Please refer to Chapter 5 for information about the proposed system controller algorithms.

Initialize the bus drivers

Before the bus drivers will function properly, they must be configured as described in Table 3.3.6. Initializing the bus drivers is simply a matter of executing the three following steps (note: order is important).

Step 1: Write \$92 to \$A003.

Step 2: Write \$80 to \$A007.

Step 3: Write \$3 to \$A004.

This initialization routine configures the data lines in the "read" direction.

Read/Write operations with the data bus

The following is a description of the sequences that must be performed during a read and write operation with a board on the data bus.

Read data from board.

Note: the first three steps listed below are optional if the addresses have all ready been set (as is the case during multiple operations to the same board).

Step 1: Set the data direction to READ (write $92 to $A003).

Step 2: Set the 4-bit board address (D_0-D_3 of $A002).

Step 3: Set the 4-bit register address (D_4-D_7 of $A002).

Step 4: Set the 16-bit address line to the memory address desired ($A005, $A006).

Step 5: Set the "bus read" line, D_1 of $A004, to active (LOW).

Step 6. Read the 16-bit data from the bus data ports (MSB from $A000 , LSB from $A001).

Step 7. Reset the "bus read" line, D_1 of $A004, to inactive (HIGH).

*** The read operation is complete. ***

Write data to board.

Note: As with the read operations, the first three steps listed below are optional if the addresses have all ready been set (as is the case during multiple operations to the same board).

Step 1: Set the data direction to WRITE (write \$80 to \$A003).

Step 2: Set the 4-bit board address (D_0-D_3 of \$A002).

Step 3: Set the 4-bit register address (D_4-D_7 of \$A002).

Step 4: Set the 16-bit address line to the memory address desired (\$A005, \$A006).

Step 5. Write the 16-bit data to the bus data ports (MSB to \$A000, LSB to \$A001).

Step 6: Set the "bus write" line, D_0 of \$A004, to active (LOW), then reset it to inactive (HIGH). If necessary, a pause may be inserted before returning the "bus write" line to inactive, though in most circumstances no pause is necessary.

*** The write operation is complete. ***

<u>The system clock</u>

As noted in Table 3.3.1, one of the system bus lines is the system clock. A BNC-connector aboard the system front-end provides a means to connect a TTL-compatible signal to the system clock line. The system-clock bus line is controlled directly by the signal when D_6 of bus port \$A004 is HIGH; otherwise, set this control bit LOW. When D_6 of \$A004 is LOW, the system bus clock either (1) floats HIGH (unaffected by the signal at the signal connected to the system clock connector, or (2) controlled by a board connected to the system bus.

The system trigger

A line on the system bus called the system trigger is useful for simultaneously triggering several boards. The trigger source for this line may be one of three places, including: (1) any of the boards connected to the bus; (2) the system controller; (3) a signal connected to the system trigger connection on the system controller board.

The operation of the system trigger line is controlled by two lines addressable from bus driver port $A004. The following is a description of the steps that must be taken when the trigger source is one the three listed above.

system trigger source: an I/O board

The configuration of the system trigger circuitry permits any open-drain device connected to the system trigger line to control it. This configuration is facilitated by the following steps.

Step 1: Set D_4 of bus port $A004 to LOW.
Step 2: Set D_5 of bus port $A004 to HIGH.

system trigger source: the system controller

Using the system controller as the system trigger source is possible by the following steps.

Note: Before performing any of the following steps, all boards must be in a "standby" mode to prevent premature triggering.

Step 1: Set D_4 of bus port $A004 to LOW.

Step 2: Set D_5 of bus port $A004 to:

LOW if activation-edge is rising edge;
HIGH if activation-edge is falling edge.

Step 3: "Arm" all boards on which triggering is desired. The trigger sensitivity on each board must be set in accordance with that selected in Step 2.

Step 4: When the trigger is desired, toggle D_5 of bus port $A004.

<u>system trigger source: external signal</u>

The third source for the system trigger is an external source. This external source is connected to the system trigger connector on the system controller board.

Note: Before performing any of the following steps, all boards must be in a "standby" mode to prevent premature triggering. In addition, the trigger source must be connected to the system trigger connection.

Step 1: Set D_4 of bus port $A004 to LOW.

Step 2: Set D_5 of bus port $A004 to:

LOW if activation-edge is rising edge;
HIGH if activation-edge is falling edge.

Step 3: "Arm" all boards on which triggering is desired. The trigger sensitivity on each board must be set in accordance with that selected in Step 2.

Step 4: Set D_4 of bus port $A004 to HIGH. The external trigger source is now connected to the bus trigger line, and the occurrence of the edge selected in Step 2 will trigger each of the boards.

3.5. The Circuits of the System Front-end: A Technician's Perspective

To facilitate the many duties of the system front-end, it was divided into four sections: the system controller, the bus driver, the bus trigger/bus clock circuitry, and the system power supply. These four sections are collectively responsible for I/O with the host computer and the digital signals and power present on the bus. Fig. 3.5.1 illustrates the arrangement of these components within the DAS front-end, and the following is a description of each component.

The system controller

A small, single board computer -- Motorola's 68HC11EVB -- was selected to serve as the system controller. The 68HC11EVB is an evaluation board for Motorola's 68HC11 8-bit microcomputer unit (MCU). This board has an on-board RS-232 communication port (for host communication) and sufficient parallel I/O to control the bus driver. This board also has sufficient memory space for the controller software [1]. Information about the system controller algorithms is given in Chapter 5.

Unfortunately, the 68HC11EVB does not have address and data lines directly available on its expansion connector. Therefore, modifications shown in Appendix E were necessary before the 68HC11EVB could be used as the

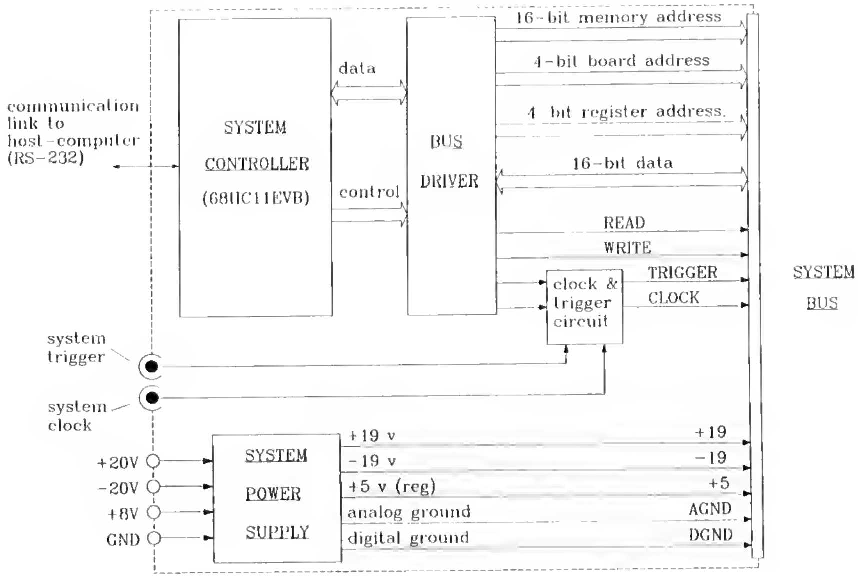


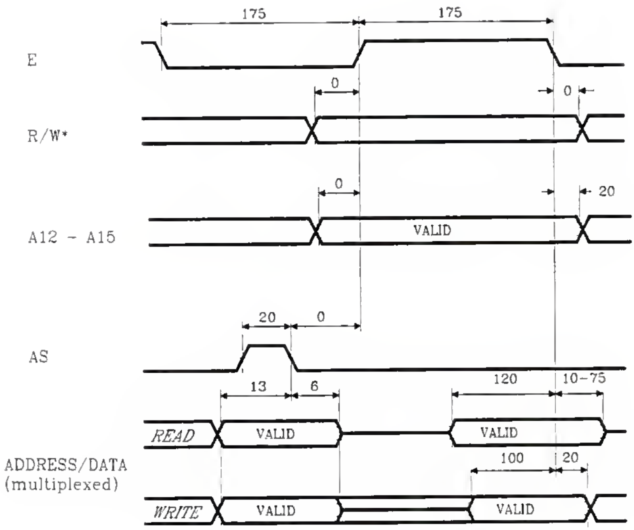
Figure 3.5.1 The components of the DAS front-end.

system controller.

The interface circuit used to join the 68HC11EVB to the bus drivers is shown in Schematic 3.1. When the 68HC11EVB is addressing memory between $A000 - $AFFF, the bus driver is enabled (this address is hard-wired via U1 and part of U2). The 3-bit port address is attained from the three least significant bits of the data when clocked by the AS line (see Fig. 3.5.2). The port-address latch (U4) is enabled when EVB_sense is pulled HIGH. An alternate method to supply the port address is to use lines marked A0, A1, and A2 -- and disable the port-address latch by pulling EVB_sense LOW. A0-A2 were used in the prototype circuit described in Appendix A.

<u>The bus drivers</u>

The bus driver is the second component of the system front-end. Fig. 3.5.3 illustrates the components of the bus driver in a block-diagram format. Each group of bus signals is supplied by their own data latch. Data I/O with the these latches is by way of the 8-bit bus driver data from the system controller. When a value is being assigned to a group, the value for the signal group is latched (or "read from" in the case of the 16-bit data lines) via the 8-bit bus driver data lines from the system controller.



Notes: 1. These drawings are NOT TO SCALE.
2. All times are in nanoseconds, and should be reguarded as minimums.

Figure 3.5.2 Timing associated with the bus driver circuit when used with the 68HC11EVB as the system controller.

Signals on the bus are controlled by the system bus driver. As shown in Schematics 3.2 and 3.3, the system bus driver is composed of a pair of 82C55A parallel peripheral interfaces (U5, U6). These devices each have three bidirectional 8-bit ports. A reset circuit (R1, C2) ensure that the bus drivers reset properly at power up. All bus lines are high-impedance from the time the system power is applied until the ports are configured otherwise. Therefore, 10 kohm pull-up resistors (RN1-RN6) were connected to each of the bus lines to protect CMOS circuitry connected to the bus that would otherwise float with a high-impedance source.

The first 82C55A (U5) bus driver is responsible for the 16-bit bus data (BD0-BD15), the 4-bit board address (BD_ADR0-BD_ADR3), and the 4-bit register address (REG_ADR0-REG_ADR3). The second 82C55A (U6) bus driver is responsible for the 16-bit memory address (BA0-BA15), and assorted bus control signals.

Bus trigger and clock control

The bus clock and bus trigger circuitry are the third component of the system front-end. Fig. 3.5.4 illustrates the duties of the system clock controller.

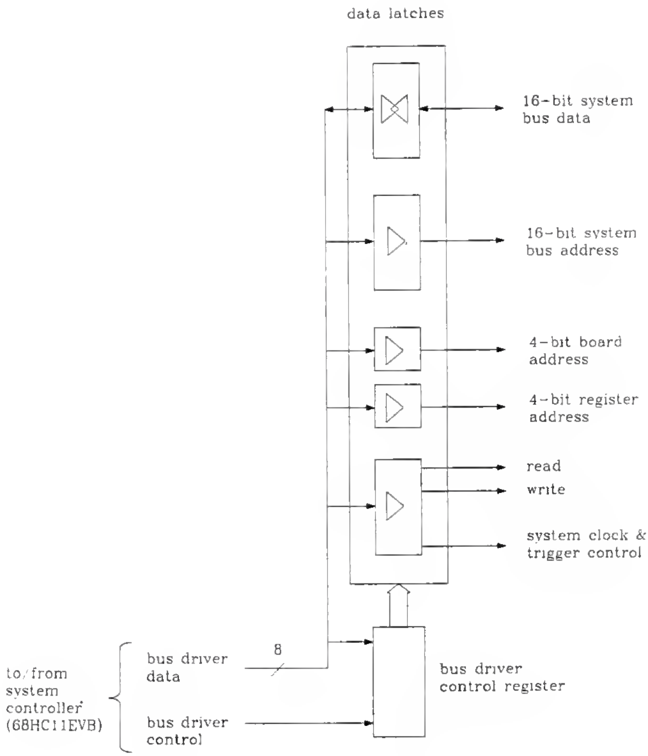


Figure 3.5.3 A block representation of the bus driver.

When the clock connected to the system clock connection is the desired system clock, this clock signal is connected to an open-drain gate which puts the signal on the bus clock line. An open-drain device was used since it enables other sources on the bus to provide the system clock (when the system clock connection is not being used).

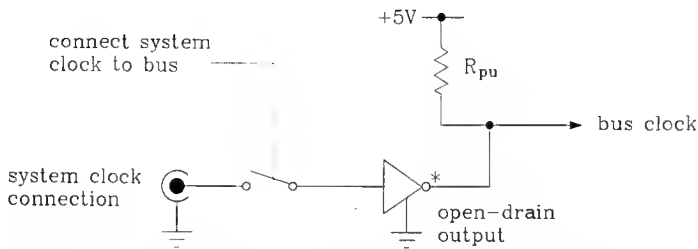


Figure 3.5.4 A block representation of the system clock controller.

Schematic 3.4(a) shows the circuitry used for the system bus clock control. An external oscillator (TTL compatible) may be used as the system clock by connecting it to the system clock BNC connector (J1). This signal is clamped by a pair of diodes (D1, D2) to protect the remainder of the circuit from improper signal amplitudes. The inverted clock signal is gated by an open-drain NAND gate (U3c) such that the external oscillator is not present on the system bus clock line, BUS_CLK, unless

SEL_EXT_CLK is active. This open-drain gate is pulled high by a 10 kohm pull-up.

A block diagram for the system bus trigger is shown in Fig. 3.5.5.The control and interface shown in Fig. 3.5.5 enables two triggers sources for the system trigger: (1) the external system trigger connection, and (2) the trigger signal from the system controller. Whichever source is selected, it is connected to the bus trigger by an open-drain gate. As with the bus clock, the bus trigger uses an open-drain gate as the bus driver to permit other boards on the bus to be the source of the bus trigger.

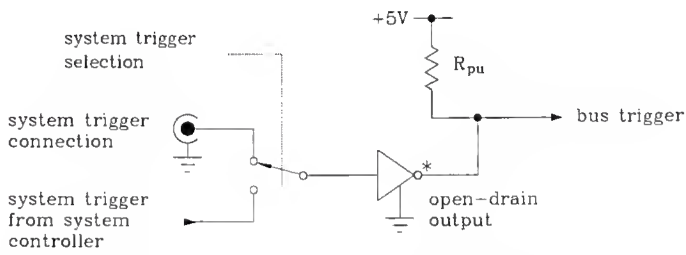


Figure 3.5.5 A block representation of the system bus trigger control and interface.

As shown in Schematic 3.4(b), a signal (TTL-compatible) to be used as the system trigger is connected to the system by way of the system trigger BNC connector (J2). As with the clock input, the system trigger input is clamped by a pair of diodes (D3, D4). A multiplexer implemented in NAND gates (U7c, U7d, U3b) selects between two trigger sources: (1) the external trigger and (2) the system controller. The external trigger source is selected when SEL_EXT_TRIG is active, and the system controller trigger line, SYS_TRIG, is selected otherwise. An open-drain NAND gate (U3d) sets the system bus trigger, TRIG_BUS, to the logic level of the selected trigger source.

System power supply

The DAS power supply is the fourth component of the system front-end. This power supply is responsible for providing power to the system bus, as well as power for the 68HC11EVB. Voltages available on the bus include +5 volts (regulated) accompanied by a digital ground, and ±18 volts (unregulated) accompanied by an analog ground. The 68HC11EVB requires ±12 and +5 volts.

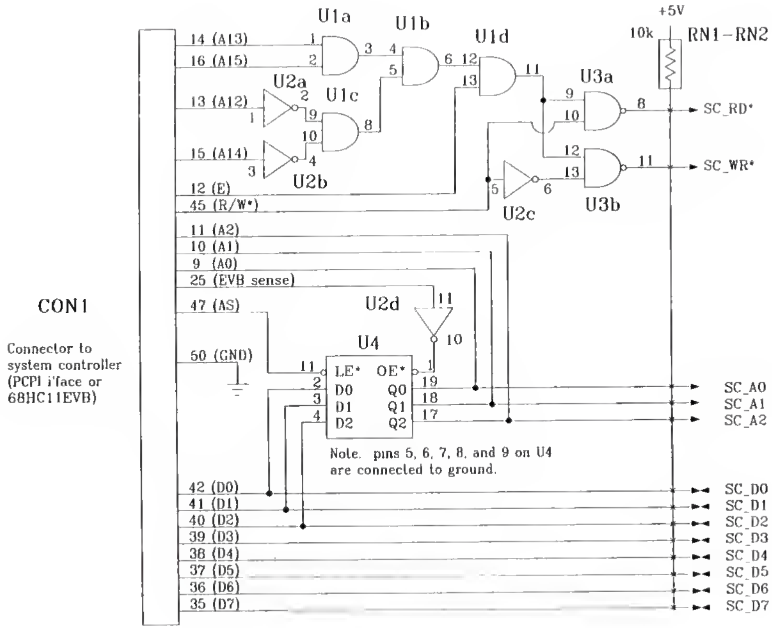
As shown in Schematic 3.5, +20 volts and -20 volts are supplied to the system via two banana jacks (J3, J4). Since the power to the system is provided by the user with banana connectors there exists a chance for reverse polarity. Therefore, a bridge rectifier (BR1) was placed between the power supply connections and the system bus to ensure proper polarity despite user negligence. The ±19 volt system bus power are the outputs of this bridge rectifier.

As mentioned, the system controller (68HC11EVB) requires three different voltages, ±12 and +5 volts. The +12 volt supply is provided by a 7812 regulator (U8), and the -12 volt supply is provided by a 7912 regulator (U9), both of which are regulated from the ±19 volt supply on the system bus. The +5 volts is provided by the system bus's +5 volt supply.

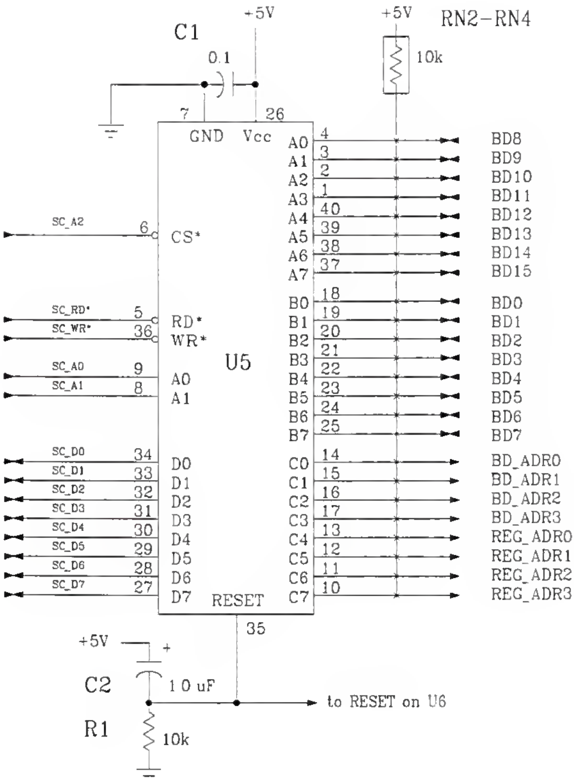
The system's +5 volt power source is supplied by an LM323 regulator (U10), which regulates an external power source (V_{in} > 8 volts) down to 5.0 volts. A crowbar circuit (for overvoltage protection) is provided on the output of the LM323. The crowbar circuit is composed of a 2-amp fuse (F1), a 5.6 volt zener diode (D5), and a sensitive-gate SCR (SCR1). When the voltage at the output of the LM323 exceeds 5.6 volts, a voltage appears at the SCR's gate, thus latching the SCR. The short circuit on the LM323's output results in a high current through the fuse which breaks the circuit.

Schematics of the front-end circuitry

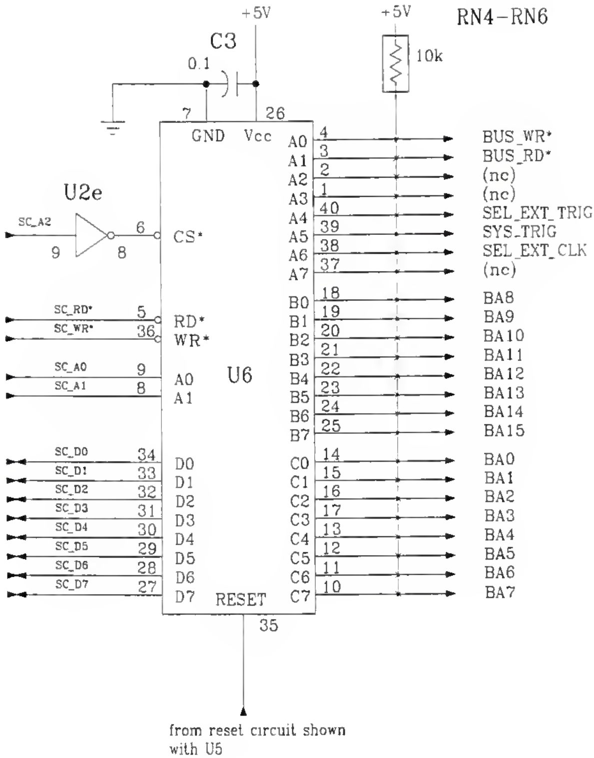
The circuits which make up the system front-end are shown in the following schematics. Also, the connections to the bus are shown in Table 3.5.1. Following the schematics is the parts list for these circuits.



Schematic 3.1 The system controller interface circuit.

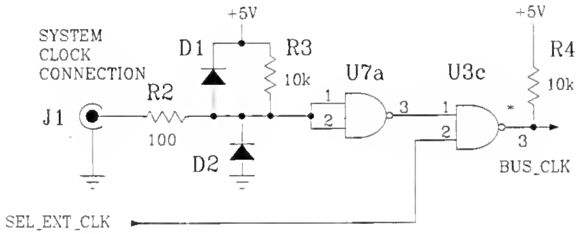


Schematic 3.2 The bus driver for data and register/board addresses.

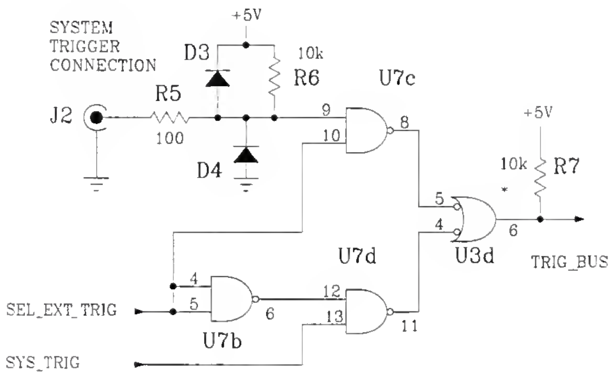


Schematic 3.3 The bus driver for 16-bit memory addresses and bus control.

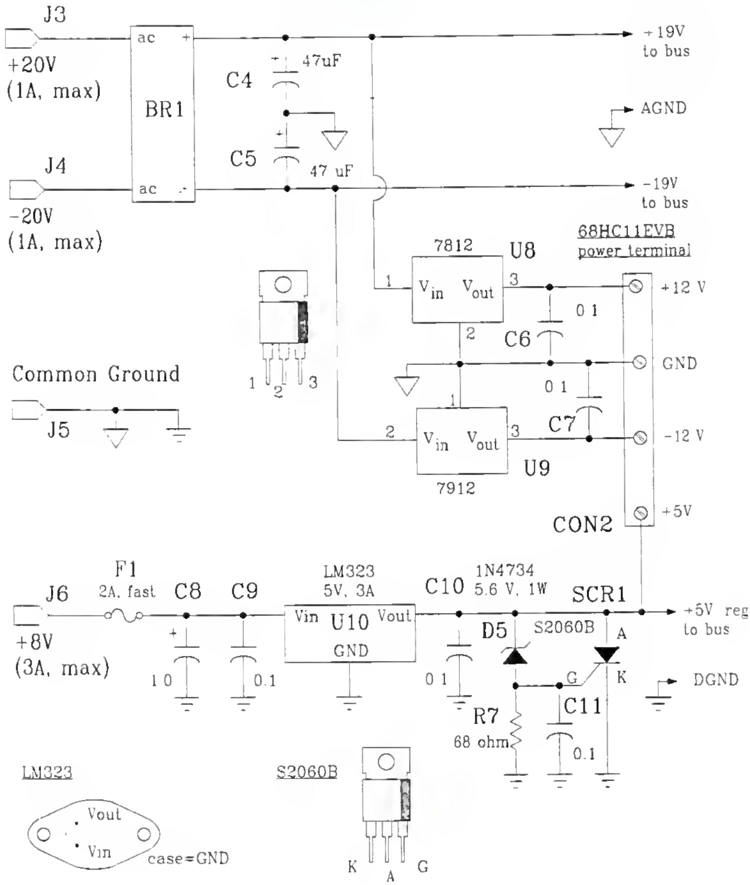
(a) the system bus clock circuit



(b) the system bus trigger circuit



Schematic 3.4 Interface and control for (a) the system bus clock, and (b) the system bus trigger.



Schematic 3.5 System power supply conditioning and regulation.

Table 3.5.1 The pin out of the system bus, as viewed from the connecter edge.

DGND	1	72	DGND
AGND	2	71	AGND
+19V	3	70	+19V
AGND	4	69	AGND
-19V	5	68	-19V
AGND	6	67	AGND
DGND	7	66	DGND
BD_ADR0	8	65	BD_ADR1
BD_ADR2	9	64	BD_ADR3
REG_ADR0	10	63	REG_ADR1
REG_ADR2	11	62	REG_ADR3
BA0	12	61	BA1
BA2	13	60	BA3
BA4	14	59	BA5
BA6	15	58	BA7
BA8	16	57	BA9
BA10	17	56	BA11
BA12	18	55	BA13
BA14	19	54	BA15
DGND	20	53	DGND
BD0	21	52	BD1
BD2	22	51	BD2
BD4	23	50	BD4
BD6	24	49	BD6
BD8	25	48	BD8
BD10	26	47	BD10
BD12	27	46	BD12
BD14	28	45	BD14
DGND	29	44	DGND
BUS_CLK*	30	43	(reserved)
RD*	31	42	(reserved)
WR*	32	41	BUS_TRIG*
DGND	33	40	DGND
+5V (REG)	34	39	+5V (REG)
+5V (REG)	35	38	+5V (REG)
DGND	36	37	DGND

Parts list for System Controller Board

BR1	100 PRV	Bridge Rectifier, 2A
C1	0.1 uF	monolithic capacitor, 50V
C2	1.0 uF	tantalum, 35V
C3	0.1 uF	monolithic capacitor, 50V
C4,C5	47 uF	electrolytic capacitor, 35V
C6,C7	0.1 uF	monolithic capacitor, 50V
C11-C13	0.1 uF	monolithic capacitor, 50V
D1-D4	1N4148	switching diode
D5	1N4734	5.6V zener diode, 1W
F1	Fuse, 2A fast-blow	
J1,J2	BNC (chassis-mount female)	
J3-J6	banana (female)	

(all resistors 1/4 W, 5% unless otherwise noted)

R1	100 ohm
R2,R3	10 kohm
R4	100 ohm
R5,R6	10 kohm
R7	68 ohm
RN1-RN6	10 kohm x 9 resistor network
SCR1	S2060B sensitive-gate SCR, 4A
U1	74HC08 quad 2-input AND
U2	74HC04 hex inverter
U3	74HC03 quad 2-input NAND, open-drain
U4	74HC573 octal latch
U5,U6	82C55A PPI
U7	74HC00 quad 2-input NAND
U8	7812, +12V regulator, 1A
U9	7912, -12V regulator, 1A
U10	LM323, +5V regulator, 3A

CHAPTER FOUR

THE ANALOG-TO-DIGITAL BOARD

4.1. Introduction

One of the design goals of the DAS project was to design and construct an I/O board suitable for analog-to-digital conversion. The purpose of this chapter is to describe the board designed for this purpose: the A/D board. The specifications for the board described in this chapter are presented in Table 4.1.1.

The presentation of the A/D board is divided into several sections. The first section provides a user's perspective of the A/D board. The second section gives information pertaining to programming the A/D board: a programmer's perspective. The third section presents the circuits that compose the A/D board: a technician's perspective.

Table 4.1.1 Specifications for the A/D board.

SIGNAL-RELATED SPECIFICATIONS

signal type:	bi-polar differential
maximum signal amplitude (gain = 1):	± 5.000 volts
Input:	
resistance	10^9 ohm
capacitance	10 pF
gain selections:	1, 10, 100, 200, 500
Bandwidth limitations (small signal = -3dB)	100 kHz (gain = 200) 50 kHz (gain = 500)
CMRR:	>120 dB @ 60 Hz

additional signal-related features:

* on-board programmable anti-aliasing filter (the Crystal CS7008 8th-order switched-capacitor filter)

* signal input overload protection

ANALOG-TO-DIGITAL CONVERSION

resolution:	12-bit, 2's compliment
max. conversion rate:	150,000 samples/second

CONVERSION CONTROL

three trigger sources:

signal trigger:	signal-level sensitive, where the trigger level is front-panel adjustable. Trigger may be configured to occur when signal is negative- or positive-going.
bus trigger:	edge-sensitive (rising or falling)
front-panel trigger:	TTL-compatible (input is protected); edge-sensitive (rising or falling)

Table 4.1.1 Specifications for the A/D board (cont).

CONVERSION CONTROL (cont.)

three clock sources:

bus clock:	conversion occurs on the falling edge of the bus clock; the frequency of the bus clock must not exceed 150 kHz.
front panel clock:	TTL-compatible (input is protected); conversion occurs on the falling edge of the clock; the frequency of the front panel clock must not exceed 150 kHz.
on-board clock:	selectable from two clocks: one for conversion rates 0.2 - 100 Hz, and the other for rates 100 - 150 kHz. Each clock is adjustable by way of a 16-bit binary counter.
conversion modes:	convert immediately, convert on trigger, convert on trigger with pre-trigger sample retention

ON-BOARD MEMORY

memory size: 64K x 16-bit

(Note: The sum of pre-trigger and post-trigger samples may not exceed 64K.)

BUS INTERFACE

Compatible with the DAS interface described in chapter 3.

POWER REQUIREMENTS

+5V,	regulated,	1A	maximum;
+19V,	unregulated,	600 mA	maximum;
-19V,	unregulated,	600 mA	maximum.

4.2. The User's Perspective of the A/D Board

The A/D board provides analog-to-digital conversion for the DAS described in Chapter 3. This board encompasses all of the features listed in Table 4.1.1. Using the A/D board is simply a matter of installing the board in an unused bus connector and commanding the board by way of commands issued by the host computer. A suggested list of commands for the A/D board are listed in Appendix B.

Care of the A/D board

Before installing the A/D board, please consult Chapter 3 for instructions regarding board insertion and removal. The A/D board is precision electronic equipment and should be handled with care at all times, especially when installing and removing it from the system front-end. When the A/D board is not in use, it should be removed from the system and stored in a safe place, preferably in an anti-static package.

A/D board front panel description

A survey of the A/D board's front panel, shown in Fig. 4.2.1, reveals many of the board's features. The components of the A/D board front panel are described as follows.

ADJ1 A triggering option is "signal level", and ADJ1 provides an adjustment for this level. The

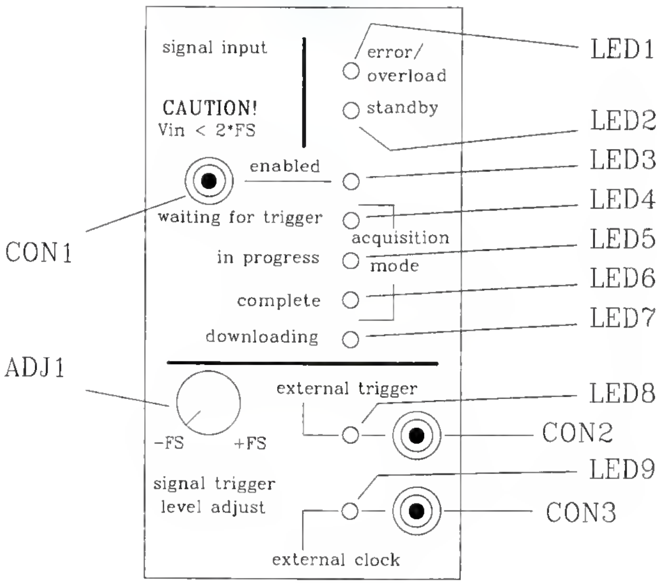


Figure 4.2.1 The front panel of the A/D board.

signal trigger level is variable from the most negative (-FS) to the most positive (+FS) measurement in the converter's range.

CON1 The signal which A/D conversions are to be made is connected to the isolated BNC connector, CON1. To obtain conversions of the highest caliber, use high-quality cables suitable for differential measurements.

CON2	External TTL-level trigger sources unique to the A/D board are connected to the BNC connector, CON2. The ground of this connector is common to the system digital ground.
CON3	The external TTL-level clock sources are connected to the BNC connector, CON3. As with CON2, this connector's ground is connected to the system digital ground.
LED1	Illumination of LED1 indicates one of two things: the board is not properly configured, or the signal at CON1 has overloaded the board's conversion circuit. If this LED illuminates while data is being acquired, the user must reset the A/D board from the host computer before conversion will continue.
LED2	LED2 indicates the board is in a non-conversion mode.
LED3	This LED illuminates when CON1 is enabled i.e. the signal present at CON1 is coupled to the conversion circuitry.
LED4	LED4 indicates the board is "armed" i.e. waiting for a trigger. This LED extinguishes once the trigger is received.
LED5	The acquisition sequence is in progress while LED5 is illuminated.
LED6	LED6 illuminates when the acquisition sequence is complete.
LED7	This LED flickers while data is being retrieved from the A/D board's memory.

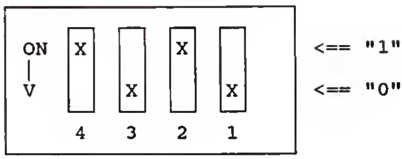
LED8	LED8 illuminates when the trigger source is the external trigger connected to CON2.
LED9	LED9 illuminates when the clock connected to CON3 is the conversion clock for the board.

The effects of circuit protection on data acquisition

The amplifier to which the external signal is connected is very sensitive to overloads, and is therefore equipped with an overload sensor. When the sensor is activated, the signal is disconnected from the amplifier and ceases the acquisition sequence. This disconnection occurs when the external signal's magnitude exceeds TWICE the full scale range. If a signal has a good deal of transients, the sensor may be deactivated (via software control), however, this leaves the input amplifier completely at the mercy of the user.

Setting the board address of the A/D board

The board address of the A/D board is determined by the setting of the 4-station DIP switch on the A/D board. Fig. 4.2.2 illustrates the orientation of the board



Switch	Value
1	2^0
2	2^1
3	2^2
4	2^3

As shown, the address is:

$2^3 + 2^1 = 10$ dec.

Figure 4.2.2 The A/D board address selection switch.

address switch. As shown in Fig. 4.2.2, if the switch is in the "off" position, that bit is counted (see example in Fig. 4.2.2). As mentioned in Chapter 3, it is very important that no two boards installed in the system share the same board address, as this would result in damage to the boards.

4.3. The A/D from a Programmer's Perspective

The purpose of this section is to present various aspects of programming the A/D board. This includes the A/D board register set and considerations that should be made when composing control algorithms.

4.3.1. The A/D Board Register Set

The A/D board is controlled and monitored by register-oriented operations. The purpose of this section is to describe each of the A/D board's registers and their function.

Like all other boards compatible with the DAS, I/O between the system controller and the A/D board is via register operations. The A/D board has ten of sixteen possible registers. Table 4.3.1.1 lists the A/D board registers and their address. The description of each of the A/D board registers follows.

Table 4.3.1.1 Register assignments for the A/D board.

Address	Register Name	Data Direction
0	A/D board EPROM and board presence indicator.*	read
1	On-board sample memory	read/write
2	Counter register	read/write
3	Board status	read
4	Board control #1	read/write
5	Trigger address	read
6	On-board programmable filter	read/write
7	A/D single conversion register	read/write
11	Control/status chip configuration register	read/write
12	Board control #2	read/write

* This register is reserved on all boards for the board's EPROM and board presence indicator.

Register 0: A/D board EPROM and board presence indicator

In accordance with the system specifications, register 0 has two purposes. First, the presence of the board can be verified by reading the most significant bit: a LOW means a board is present. Second, reading the least

Table 4.3.1.2 Bit-wise assignment of register 0.

Data Bits	Description
D_0-D_7	EPROM data
D_8-D_{14}	(reserved)
D_{15}	board presence indicator ("0" = "board present")

significant byte retrieves data from the board's program EPROM, where the controlling programs for the A/D board are stored. The bit-wise assignment of register 0 is given in Table 4.3.1.2.

To verify presence of the board:

1) read the most significant byte (D_8-D_{15}) of register 0;

2) if D_{15} is LOW ("0"), the board is present.

To load A/D board controlling programs from the board's EPROM:

1) set the "bus memory address" to the memory address that is to be read;

2) read the EPROM data--the least significant byte (D_0-D_7) of register 0.

3) repeat steps 1 and 2 until the complete set of control programs has been read.

<u>Register 1: On-board memory access</u>

The purpose of on-board memory is to store acquired samples during the acquisition sequence until they can be retrieved by the host. Retrieval of the samples is accomplished by reading the sample data from register 1. Additionally, on-board memory may be written to via register 1. This permits other boards to utilize the A/D board's memory, and it enables the on-board memory to be tested. The bit-wise assignment of register 1 is given in Table 4.3.1.3.

Table 4.3.1.3 The bit-wise assignment of register 1.

Data Bits	Description
D_0-D_{15}	On-board memory Note: When A/D converter data is stored in D_0-D_{15}; D_0-D_3 are always "0".

The 16-bit "memory address lines" on the bus set the address of the memory that register 1 is accessing.

NOTE: ATTEMPTS TO WRITE TO REGISTER 1 WHILE THE BOARD IS IN A NON-STANDBY MODE ARE IGNORED BY THE A/D BOARD.

<u>Register 2: interval timer/post-trigger sample counter</u>

Three 16-bit counters are used on the A/D board, and control of each is accessible via register 2. Register 2 is actually a multi-purpose register, where the counter being written to (or read from) is determined by "counter selector" found on control #2 (register 12). The bit-wise data assignment for register 2 is given in Table 4.3.1.4.

The following section details the procedure for reading from or writing to a counter via register 2.

1) Set the "counter selector" on control #2 to the "counter control register".

Table 4.3.1.4 The bit-wise assignment of register 2.

Data Bits	Description
D_0-D_7	(reserved)
D_8-D_{15}	8-bit counter data

2) Depending upon which counter is being addressed, write the appropriate value to register 2 given in the following list:

counter name	write value
sample period generator	$14
digital one-shot	$52
post-trigger sample counter	$90

3) Set the "counter selector" on control #2 to the counter being addressed.

4) Write (read) the least-significant-byte (LSB) of the counter to (from) register 2.

5) Set the "counter selector" on control #2 to the "counter control register".

6) Depending upon which counter is being addressed, write the appropriate value to register 2 given in the following list:

counter name	write value
sample period generator	$24
digital one-shot	$62
post-trigger sample counter	$A0

7) Set the "counter selector" on control #2 to the counter being addressed.

8) Write (read) the most-significant-byte (MSB) of the counter to (from) register 2.

Each counter has a unique function. The 16-bit value written to a counter is determined in the following procedures.

1. Sample Period Generator

The purpose of the sample period generator is to divide the A/D board's internal clock to provide a desired sampling frequency. The 16-bit binary value written to the sample period generator is the closest integer given by

$$\text{value} = \frac{f_{osc}}{DIV * f_{samp}} \qquad \text{...(4.1)}$$

where f_{osc} is the frequency of the on-board oscillator, DIV is the oscillator pre-divider, and f_{samp} is the desired sampling rate, in Hertz. The value for DIV in Eq. 4.1 is determined by the clock selected in control #1, where DIV is given by

Clock selected	DIV
internal, hi-speed	2
internal, lo-speed	1024

2. Digital one-shot

The purpose of the digital one-shot is to generate a pulse for the conversion circuitry. The width of this pulse is controlled by this counter, which is given by

$$t_{pulse} = \frac{n}{5 * 10^6} \quad \text{(seconds)}$$

where n is the 16-bit binary value stored in the one-shot counter, and t_{pulse} is the pulse width, in seconds. The value for this counter is usually assigned when the A/D board is initialized.

3. Post-trigger sample counter

The purpose of this counter is to count the number of conversions that are made, and to stop the conversion sequence when the requested number of samples have been acquired. Therefore, the value assigned to this counter is given by

$$N = \text{Samples} - 1$$

where Samples is the number of post-trigger samples, and **N** is the number assigned to the counter.

Register 3: status

Information concerning the status of the A/D board may be obtained by reading register 3 and comparing the contents of the register with the bit-level descriptions as described in Table 4.3.1.5.

Table 4.3.1.5 Assignments for the A/D board status register.

BIT	SIGNAL	ACTIVE	SIGNAL DESCRIPTION
D_0	bd_error*	LOW	this signal becomes active when: 1) power is first applied to the board, 2) power is absent from the board, 3) signal input is severely overloaded.
D_1	seq_end*	LOW	this signal becomes active when the acquisition sequence is completed i.e. the desired number of samples have been acquired.

D_2	trig_recvd*	LOW	this signal becomes active when the trigger has been received. The trigger selection parameters are determined by the control registers.
D_3	mem_cycle*	LOW	This signal becomes active when the on-board memory has recycled while waiting for the trigger. This is useful for determining the starting address of a pre-trigger acquisition data sequence.
D_4	a/d_eoc	HIGH	This signal is a duplicate of the analog-to-digital (A/D) converter status bit. This signal is useful when data acquisition is managed by an external controller. The A/D is finished converting when this signal is active.
D_5- D_7	unused	n/a	Reserved.

Register 4: control #1

Register 4 is the first of two control registers on the A/D board (control #2 is register 12). As the name implies, the contents written to this register control various aspects of the A/D board. The bit-wise assignment for register 4 is given in Table 4.3.1.6. Follows is a description of each of the controls accessible via control #1.

Table 4.3.1.6 Bit-wise assignment for register 4.

Data Bits	Description
D_0-D_7	Control #1 data (register 4)
D_8-D_{15}	(reserved)

Conversion Mode

Value D_1	D_0	Description
0	0	standby (no conversion activity)
0	1	convert immediately
1	0	convert on trigger (no pre-trigger acquisition)
1	1	convert on trigger (with pre-trigger acquisition)

On reset, the board mode is set to standby.

Trigger selection

Value D_3	D_2	Description
0	0	select SIGNAL trigger input for trigger source
0	1	select SIGNAL trigger input for trigger source -AND- pull down bus trigger simultaneously
1	0	select BUS trigger input for trigger source
1	1	select PANEL trigger for trigger source

On reset, the trigger mode is set to SIGNAL trigger.

Trigger edge selection

If a trigger dependent conversion mode is selected (via board mode) this bit selects the appropriate edge on which conversion will occur.

Value D_4	Description
0	trigger on POSITIVE GOING edge
1	trigger on NEGATIVE GOING edge

On reset, the trigger edge is set to POSITIVE going edge.

Clock source selection

Value D_6	D_5	Description
0	0	internal, hi-speed (100 Hz - 150 kHz sampling)
0	1	internal, low-speed (0.18 Hz - 100 Hz sampling)
1	0	Bus clock.
1	1	Front panel clock.

On reset, the clock select is set to internal hi-speed clock.

Filter enable

This control bit determines whether or not the on-board anti-aliasing filter is in the analog signal path. Removal of the anti-aliasing filter may be desireable if an external filter is being used.

Value D_7	Description
0	filter is removed from signal path
1	filter is included in signal path

On reset, the filter is removed from the analog signal path.

Register 5: trigger address

The 16-bit address of the sample acquired when the trigger occurred is determined by reading register 5. Retrieval of the contents of this register permits the address of the beginning of the pre-trigger samples to be calculated.

Register 6: on-board filter

Register 6 is the on-board anti-aliasing filter configuration register. Programming the filter requires the placement of coefficients in the filter's configuration memory. Details for programming the filter are given in [2]. The on-board filter configuration memory may be both written to and read from. Reading the filter configuration registers is useful for verifying the receipt of the coefficients. The bit-wise data assignment for register 2 is given in Table 4.3.1.7.

Table 4.3.1.7 The bit-wise assignment of register 6.

Data Bits	Description
D_0-D_7	(reserved)
D_8-D_{13}	6-bit filter coefficients
D_{14}-D_{15}	(reserved)

The address to which the coefficients are written is determined by the six least-significant-bits of the 16-bit bus address lines, A_0 - A_5.

Register 7: A/D single conversion register

The purpose of register 7 is to permit immediate data conversion by the A/D converter (while the A/D board is in the standby mode). The procedure is as follows:

1. Initiate an analog-to-digital conversion by writing to register 7 (value written is unimportant). This causes a conversion to occur.
2. When the end-of-conversion flag (present in the status register) becomes active, the converted data may be retrieved by reading register 7. The A/D conversion value is a 2's compliment, 12-bit value spanning the full 16-bit data lines, D_0-D_{15}, where the sign bit is D_{15}. D_0-D_3 are always zero.

Register 11: Control/status register configuration

The control registers and status register are all elements of a common electrical component on the A/D board: an 82C55A Parallel Peripheral Interface. The nature of the 82C55A necessitates that when it is powered-up, it must be configured before the control registers are operational. Register 11 is the register to which the configuration control word is written, and the bit-wise arrangement of this register is given in Table 4.3.1.8.

Table 4.3.1.8 Bit-wise assignment for register 11.

Data Bits	Description
D_0-D_7	configuration register
D_8-D_{15}	(reserved)

The configuration control word for the control/status register is $82. This value is in accordance with information found in [3].

<u>Register 12: control #2</u>

Register 12 is the second of two control registers on the A/D board (control #1 is register 4). As with control #1, the contents written to this register control various aspects of the A/D board. The bit-wise assignment for register 12 is given in Table 4.3.1.9. Follows is a description of each of the controls accessible via control #2.

Table 4.3.1.9 Bit-wise assignment for register 12.

Data Bits	Description
D_0-D_7	Control #2 data (register 12)
D_8-D_{15}	(reserved)

Signal source selection

Value D_1	D_0	Description
0	0	external signal removed, instrumentation amplifier's inputs are shorted
0	1	external signal
1	0	(reserved)
1	1	signal trigger level

On reset, the external signal is removed.

Gain selection (control register bits 2, 3, 4)

Value D_4	D_3	D_2	Description
0	0	0	gain = 1 (± 5.000 V)
0	0	1	gain = 10 (± 500 mV)
0	1	0	gain = 100 (± 50 mV)
0	1	1	gain = 200 (± 25 mV)
1	0	0	gain = 500 (± 10 mV)

On reset, the gain is set to 1.

Counter selector for register 2

Value D_6	D_5	Register 2 description
0	0	sample period generator (counter 0)
0	1	digital one-shot (counter 1)
1	0	post-trigger sample counter (counter 2)
1	1	counter control register

On reset, register 2 is the sample interval timer.

Overload protection enable/disable

This control bit determines whether or not the overload protection circuit is activated. This protection should always be in place except when occasional transients are known to be of short duration.

Value D_7	Description
0	protection circuit activated
1	protection circuit defeated

On reset, the protection circuit is activated.

4.3.2. Algorithmic Control of the A/D Board

This description of controlling the A/D board is divided into four sections: initializing the board, pre-acquisition set-up, acquisition of data, and retrieval of data.

Initializing the A/D board

The A/D board must be initialized prior to its use. This initialization consists of the following steps:

1) Write the configuration value register 11 ($82).

2) Switch the conversion mode (control #1) to "convert immediately" and switch back to "standby". This clears the board error status.

3) Load the counters with the following values:

counter	initial value
sample period generator	2500 (dec)
one-shot	4
sample counter	1000 (dec)

This sets the board for 1000 samples at a sampling rate of 1 kHz, and an 800 ns conversion pulse width.

Following this initialization, the board is ready to be set-up for data acquisition.

Pre-acquisition set-up

There are numerous controls that may be adjusted prior to an acquisition sequence. Table 4.3.2.1 lists the controls and identifies the control register on which they are located. Another pre-acquisition control which must be set is the pre-trigger sample counter. In addition, if the on-board anti-aliasing filter must be programmed if it is to be used.

Table 4.3.2.1 A summary of pre-acquisition controls.

pre-acquisition control	control register
on-board filter enable/disable	1
sampling clock	1
overload protection circuit enable/disable	2
signal gain	2
signal selection = "external"	2

<u>Acquisition of data</u>

Data may be acquired in four different modes. These four modes are presented and described in Table 4.3.2.2.

Table 4.3.2.2 The data acquisition modes for the A/D board.

conversion mode	description
immediate	acquire the number of samples specified by the sample counter immediately
on-trigger	same as immediate, except acquisition begins at the receipt of trigger from the selected source
on-trigger w/ pre-trigger	acquisition begins immediately and does not stop until the number of post-trigger samples equals the number of samples specified by the sample counter
single sample	one conversion is made

The A/D board's conversion mode is established by the value written to the "conversion mode" field of control register #1.

The status of the acquisition sequence is indicated by two status bits in the status register. If the conversion mode is trigger dependent, the "trig_recvd*" status bit becomes active when the trigger is received. When the "samp_ser_end*" status bit becomes active, the acquisition sequence is completed--the data is ready to be retrieved.

<u>retrieval of data</u>

The final step of the conversion process is the retrieval of the acquired data. Retrieving the data is simply a matter of reading the data directly from the A/D board's on-board memory via register 1. The steps required for data retrieval are as follows:

1. set the board conversion mode to "standby";
2. determine the starting address for the samples (this procedure is described following this list);
3. set the 16-bit address on the bus to the starting address;
4. read register 1 . . . this 16-bit number is the sample value;
5. increment the 16-bit address on the bus and repeat step 4 until the total number of samples requested have been retrieved.

If the conversion mode was "immediate" or "trigger (not with pre-trigger)", the starting address is always 0001. Calculating the starting address for the pre-trigger acquisition mode requires a few more steps. Fig. 4.3.2.1 provides an illustrative example of the three different situations that can occur when acquiring pre-trigger samples. As shown in Fig. 4.3.2.1, samples acquired prior to the receipt of the trigger are stored sequentially throughout the sample memory e.g. samples are stored in (hexadecimal addresses) FFFE, FFFF, 0, 1 . . . .. The acquisition sequence ends after the number of post-trigger samples have been acquired following the receipt of the trigger. The procedure for calculating the starting address for the pre-trigger acquisition mode is done in the following sequence:

1) retrieve the memory address of the sample that was acquired at the same time the trigger was received (read this value from register 5 and call this value trig_addr);

2) Call the number of pre-trigger samples requested num_pre_trig;

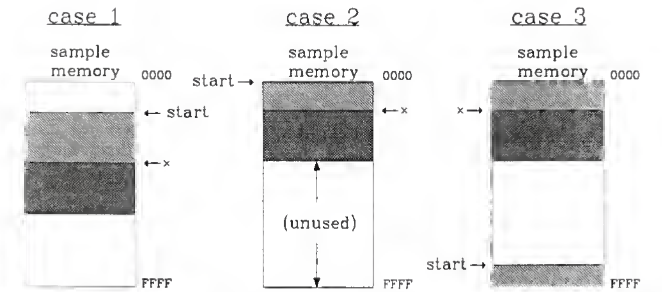
IF trig_addr > num_pre_trig

THEN, from Fig. 4.3.2.1, case 1 has occurred. The starting address of the first pre-trigger sample is given by

If the number of samples requested is

pre-trigger = 1000
post-trigger = 1000

one of these three conditions can occur:



case 1	case 2	case 3
memory cycle = ?	memory cycle = NO	memory cycle = YES
pre-trigger = 1000	pre-trigger = 500	pre-trigger = 1000
post-trigger = 1000	post-trigger = 1000	post-trigger = 1000

pre-trigger samples	post-trigger samples
x→	location where trigger occured
memory cycle	is a condition flag that indicates the sample memory has recycled while waiting for the trigger.
start	is the place where pre-trigger samples begin.

Figure 4.3.2.1 Examples of data storage during pre-trigger sample retention.

addr_pre_trig = trig_addr - num_pre_trig

where addr_pre_trig is the address of the first pre-trigger sample.

IF trig_addr < num_pre_trig

THEN, from Fig. 4.3.2.1, case 2 or 3 has occurred. First, determine if memory has cycled by examining the status register's memory cycle flag (D_3 of register 3).

If memory has NOT cycled (D_3 = INACTIVE), case 2 has occurred, and

addr_pre_trig = 0001.

Note that the actual number of pre-trigger samples is NOT the number requested, but is instead given by

num_pre_trig = trig_addr.

Otherwise, if memory has cycled (D_3 = ACTIVE), case 3 has occurred, and the starting address of the pre-trigger samples is given by

addr_pre_trig = trig_addr - num_pre_trig.

The single-sample conversion mode

A special mode of conversion is the single-sample conversion mode. An acquisition of a single sample is useful for several purposes, including: retrieval of signal-trigger level, obtaining an offset measurement for the analog circuit, and A/D conversions requested by another board. A single sample is acquired in the following manner:

1. Verify A/D board conversion mode is "standby".

2. Select appropriate signal for conversion. The signal source for the conversion is determined by the "signal route" control bits. Therefore, the route must be set according to the purpose of the single sample conversion. Table 4.3.2.3 presents the appropriate signal source for the different acquisition purposes.

Table 4.3.2.3 Signal sources appropriate for single conversions.

purpose of acquisition	route*
convert trigger level	trigger level
obtain offset	ext. signal removed
convert signal	ext. signal

*Route refers to "signal route" field, control register #2.

3. Initiate the conversion. As given in the register 7 description, a "write" to register 7 while the A/D board is in "standby" conversion mode results in a single conversion.

4. Retrieve converted data. When the status bit "A/D_EOC" becomes active, the data may be retrieved by reading the 16-bit data from register 7, where the sign bit is D_{15}, and D_0 - D_3 are zeros.

4.4 The Circuitry for the A/D Board

The A/D board is composed of many circuits that enable it to fulfill the specifications listed earlier in this chapter. Fig. 4.4.1 presents a block diagram perspective of the board, on which the major circuit elements are identified.

As shown in Fig. 4.4.1, the A/D board circuity is divided into two sections: the signal handling components (across the top), and the digital control components.

Fig. 4.4.1 is described as follows. The signal is input to an instrumentation amplifier, filtered, then undergoes 12-bit analog-to-digital conversion. This data is routed through the data source selector and retained in the on-board memory. Conversions are directed by the trigger-conversion control section. This section monitors the selected trigger source, the selected clock source, and also counts the number of conversions performed. The A/D board is controlled by writing appropriate control values to the control register; the operating status of the A/D board is determined by reading the status register. In addition, the A/D board has its own memory address generator which permits it to acquire data completely independent of the system bus.

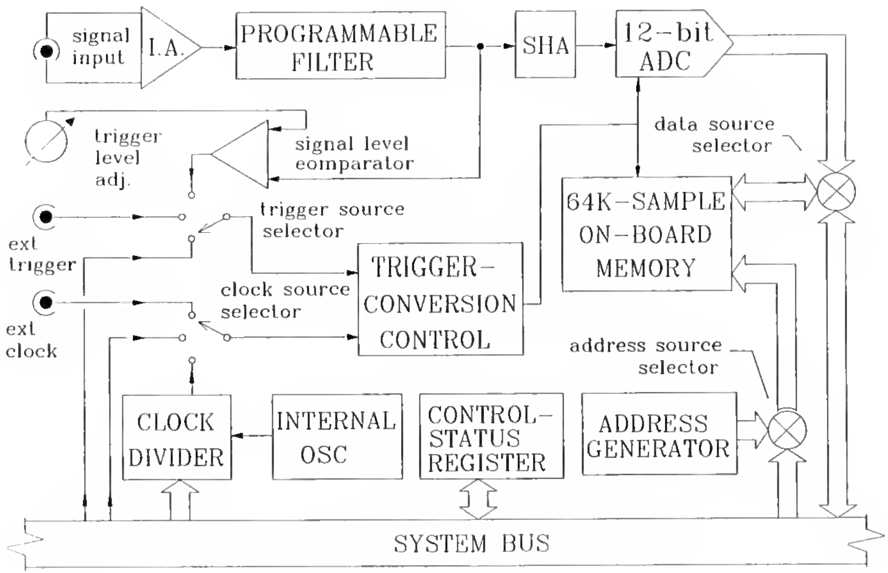


Figure 4.4.1 Block diagram of the A/D board.

The A/D board circuitry is presented throughout the remainder of this chapter in the following manner. First, the design of the analog signal handling circuitry and the analog-to-digital conversion circuitry is presented. Second, the digital circuits that control the conversion circuitry are presented. Included with the digital circuits is the circuity used to interface the A/D board to the system bus.

The end of this chapter (section 4.4.4) contains the schematics and other relevant information about the circuits. This information includes parts lists and descriptions of labels used in the schematics. Specific information about the construction of the A/D board and the parts layout are presented in Appendix D.

4.4.1. The Analog Signal Circuits

The signal-related specifications for the A/D board were obtained by the judicious selection of electrical components for the analog circuitry. The arrangement of the major analog signal components is shown in Fig. 4.4.1.1.

As shown in Fig. 4.4.1.1, the external signal is connected to an instrumentation amplifier (I.A.). The selection of the I.A. was very important to ensure the performance of the A/D board would attain the design

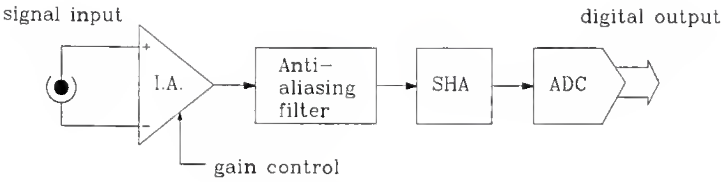


Figure 4.4.1.1 The arrangement of the major analog signal components.

specifications. Many devices were examined in catalogs with the Analog Devices AD624C being selected. This I.A. has pin-programmable gain and a common mode rejection ratio (CMRR) of 130 dB (gain=500).

An on-board programmable anti-aliasing filter was included in the design. The filter selected for this purpose was the CS7008, manufactured by Crystal Semiconductor, Corp. The CS7008 is a digitally configurable switched-capacitor filter capable of providing a filter of eighth order or below. The filter's configuration is determined by coefficients stored in the filter's configuration memory. This filter has a 72 dB dynamic range, thus making it useable in a 12-bit analog-to-digital circuit [4].

The analog-to-digital (A/D) converter selected for this circuit was Analog Devices AD578K. This A/D converter generates 12-bit 2's compliment conversion values via the successive approximation technique. This converter also has a maximum conversion time of 3.0 microseconds which permits conversions at the rate of 150 kHz.

Since the analog-to-digital converter selected for use with this circuit employs a successive approximation technique, a sample-and-hold amplifier (SHA) was used in the circuit. The important consideration when selecting a SHA for this circuit was the aperture time of the device. The aperture time is the time required for the SHA to switch from sample to hold -- a limiting factor to the overall throughput rate of the data acquisition system. The SHA selected for this circuit was the Analog Devices AD346J, which has an aperture time sufficient for 97 kHz signals being digitized with 12-bit resolution [5].

Signal routing within the analog circuit

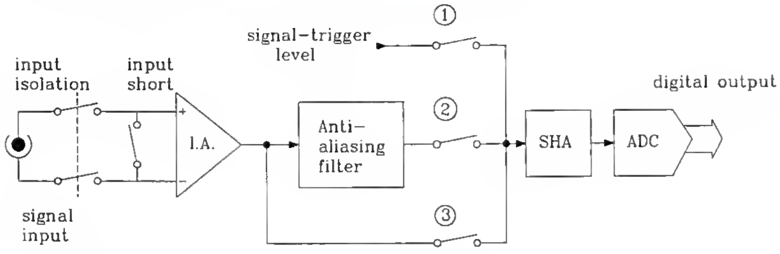


Figure 4.4.1.2 The arrangement of signal routing relays between the analog signal components.

Two features that were included on the A/D board were: (1) conversions made with the filter removed from the circuit and (2) a provision to "measure" the adjustable signal trigger level. These features were incorporated by using signal routing as depicted in Fig. 4.4.1.2.

Routing of the signals within the analog circuit was accomplished by controlling the switches as shown in Table 4.4.1.1.

Table 4.4.1.1 Signal routing logic for the analog circuit.

Signal Source	INPUT	SHORT	Switch 1	2	3
isolate	OFF	ON	ON	FIL	FIL*
signal	ON	OFF	OFF	FIL	FIL*
trig level	OFF	ON	ON	OFF	OFF
	FIL is "ON" if the filter is in-circuit.				

A method of switching the signal was needed in the analog circuit. Since the input impedance of the AD346 SHA is low (3 kohm, CMOS analog switches were not practical. Therefore, a mechanical relay was used. The relay selected for all switching applications was the Clare MSS4. This relay was selected for three reasons: 1) it has mercury-wetted contacts; 2) it is mounted in a low-profile single in-line package (SIP); 3) its solenoid is designed for 5-volt operation. The control of these relays is detailed in the digital control section (later in this chapter).

<u>The instrumentation amplifier circuit</u>

Schematic 4.1 shows the circuit details for the I.A. The differential analog signal is input by way of a BNC

connector (J100). Since the AD624C must not have a signal applied to its inputs during the absence of power supply voltage, isolation relays (REL100, 101) were placed between the signal input and the input of the I.A. These relays are the same type as those used for signal routing. Additionally, a relay (REL102) is used to short the inputs of the I.A. This shorting relay serves two functions. First, the shorting relay eliminates I.A. output drift caused by no return path for the input bias current [6]. Second, the shorting relay provides a convenient "shorted-input" reference for DC offset measurements.

Input overload protection. Unfortunately, the AD624C must be protected from signal input overload. The method suggested by Analog Devices consists of current limiting resistors in series with each input to the I.A. However, this method seriously degrades the performance of the I.A. in terms of common mode rejection and noise. Therefore, a non-intrusive input protection method was designed.

Fig. 4.4.1.3 illustrates the operation of the non-intrusive input protection circuit. The full scale conversion range of the analog circuit causes a ±5 volt swing at the output of the I.A. Therefore, when the output of the I.A. has an excursion outside ±10 volts, it is safe to assume that the inputs of the I.A. are far

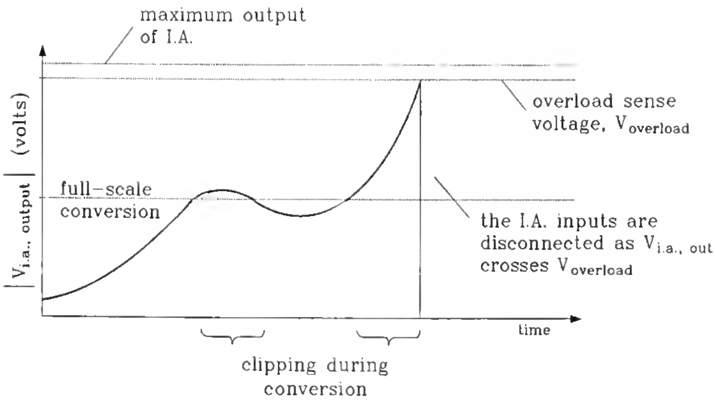


Figure 4.4.1.3 The operation of the protection circuit with an overloading signal.

enough outside the operating range and need to be disconnected. A block diagram of the protection circuit is shown in Fig. 4.4.1.4.

During normal operation the input isolation relays are controlled by the control line which either opens or closes these relays. However, when an overload condition exists, the overload comparator senses this condition and commands the input relay controller to open up the isolation relays. The relays remain open until the relay controller is reset. The user is informed of the overload condition by means of an error status LED (LED201) on the

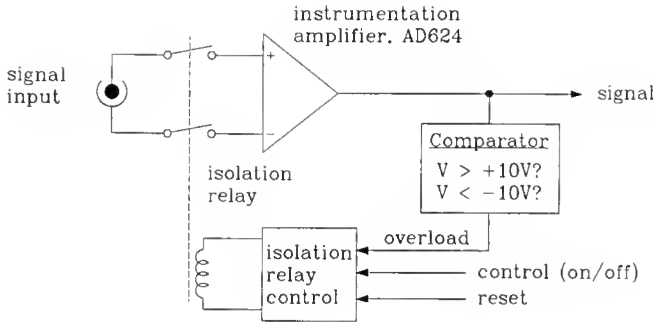


Figure 4.4.1.4 A block representation of the non-intrusive protection system used on the A/D board.

board's front panel.

The value selected for $V_{overload}$ was ±10.0 volts. This value was selected for the following reasons. The region for values of $V_{overload}$ is given by

$$V_{full\text{-}scale} < V_{overload} < V_{I.A.\ output\ rating}$$

where $V_{full\text{-}scale}$ is approximately 5.0 volts, $V_{I.A.\ output\ rating}$ is given to be 10 volts (with the power supply voltage used in this circuit), and $V_{overload}$ is the voltage at which the circuit is sufficiently overloaded to warrant the I.A. inputs being disconnected. It was important to select a value for $V_{overload}$ such that the circuit is not interrupted by signals with "normal" overloads as this would render the board useless if the signal had occasional transients. Therefore, $V_{overload}$ set to ±10.0 volts provides a sufficient overload margin but is still a value attainable by the AD624 I.A.

The overload signal comparison circuit is shown in Schematic 4.1. An LM319 dual-comparator (U101) is used as an "absolute value" magnitude comparator whose output is determined by

IF $|V_{IA,\ out}| > V_{overload}$ THEN overload* = active

where overload* is the comparator output, $V_{IA,\ out}$ is the I.A. output voltage, The +10 volt and -10 volt comparison voltages are obtained by resistive voltage dividers (R5-R8).

This protection method has two drawbacks. First, occasional transients with magnitudes greater than 2 * $V_{full\text{-}scale}$ will cause the signal inputs to be interrupted. Second, this method of protecting the amplifier is "after-the-fact" i.e. the I.A. is overloaded prior to the sensing of the overload condition, and must remain overloaded until the input isolation relays (REL100, REL101) can be disconnected (approximately 1.5 msec). However, this short duration of overload should cause no harm to the AD624 I.A. The control circuit for these relays is described later in the digital control section.

Gain Selection

Gain control for the A/D board is provided as part of the AD624 I.A. The gain of the AD624 is set by connecting a gain-select pin to one of several pins which correspond to gains of 100, 200, 500, and 1000. In addition, an extra pin is provided to permit a user-selectable gain setting (via a resistor).

The connection of the gain selection components is shown in Schematic 4.1. Relays (the same type of relays used for signal routing) are used to select the desired gain. Closing a relay selects connects the G_{select} pin of the AD624 to a corresponding gain selection pin: g=100, g=200, g=500, and G_{ext} (for externally adjustable gain). Control of these relays is described later in the description of the digital control section.

Gain = 10 is obtained by using G_{ext} in series with a external resistor. The appropriate value for gain = 10 was given by

$$\text{Gain} = 10 = 1 + \frac{40,000}{R_G} \pm 20\%$$

where gain is the gain of the AD624C I.A. and R_G is the series gain resistance, in Ohms. Solving this equation for R_G yields a value bounded by 3.6 kohm and 5.7 kohm. In the circuit, R_G is composed of a 3.65 kohm fixed resistance (R103) in series with a 2.0 kohm variable resistance (R104). Gain = 10 is attained by adjusting the variable resistance until x10 gain is obtained.

Input offset and output offset adjustment. To optimize the operation of the AD624, adjustments are available to null input and output offset bias errors. Although the circuit could be adjust for zero offset at the time of construction with fixed resistors, aging of the I.A. and changes in the I.A.'s ambient temperature will cause the offsets to change. Having both input and output offset null adjustment is important. At low gains the effects of the output offset dominate; at high gains the input offset dominates. The procedure for nulling the offset errors are presented in Table 4.4.1.2. The adjustments for input offset null (R101) and output offset null (R102) are shown in Schematic 4.1. Additional details for nulling offset errors are presented in Appendix A.

Table 4.4.1.2 Procedure for nulling I.A. (AD624) offset errors. [Source: Linear Products Databook, Analog Devices, 1988, p. 4-55.]

Input offset null adjustment	
1)	short I.A. signal input
2)	set gain to maximum (G=500)
3)	adjust the input offset nulling resistor until output equals zero.
Output offset null adjustment	
1)	set gain to 1
2)	adjust output offset nulling resistor until output equals zero.

<u>The on-board anti-aliasing filter</u>

The anti-aliasing filter selected for use with this circuit has a few limitations when used in this circuit. First, the frequency response of the filter is band-limited between 5 Hz and 20 kHz, which is somewhat lower than the design specifications for the A/D board (0-75 kHz). Therefore, this filter can only be used for "audio band" signals. Second, since the switched capacitor filter is a sampled data system, it requires two external filters: an anti-aliasing filter (at the input) and an anti-image filter (at the output). Another limitation of the CS7008 (when used in this circuit) is the maximum peak-to-peak signal amplitude (±3.0 volts) at the filter's input, thus necessitating a gain-matching network to make the filter useable.

In order to use the CS7008 filter in the A/D board analog circuit, additional circuitry adjacent to the filter was necessary as shown in Fig. 4.4.1.5.Before the signal from the instrumentation amplifier can be applied to the CS7008, it must first be passed through an anti-aliasing filter. The filter selected for this application was a 2-pole Butterworth lowpass with f_c = 20 kHz. This value for f_c was selected since the response of the CS7008 is limited to 20 kHz.

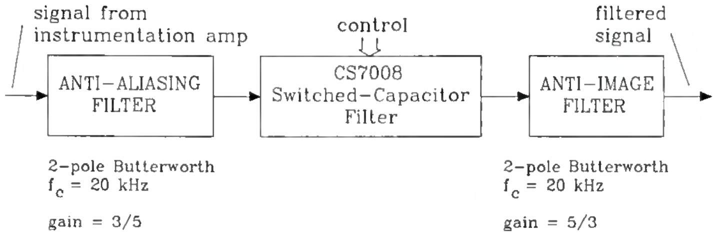


Figure 4.4.1.5 A block representation of the on-board anti-aliasing filter and its associated circuitry.

An attenuator was implemented prior to the anti-aliasing filter to match the instrumentation amplifier's full scale signal deviation to the CS7008. The gain of this stage was set at 3v / 5v, or gain = 0.6. The anti-image filter selected for this application was a filter with identical characteristics as the anti-aliasing filter. Full scale signal deviation (±5 volts) was restored following the anti-image filter with an amplifier with gain 5v / 3v, or gain = 1.67. This gain-compensation arrangement between the anti-aliasing filter and the anti-image filter makes the overall gain of this filter stage equal to the gain as given by the transfer function implemented on the CS7008.

The circuit for the on-board anti-aliasing filter is shown in Schematic 4.2. The signal from the instrumentation amplifier is attenuated (gain = 3/5) by an inverting amplifier (U102a) circuit. The output of the attenuator is clamped by a 4-volt signal clamp composed of two 3.6-volt zener diodes (D101, D102) in series with switching diodes (D103, D104). This clamp is necessary to protect the CS7008 from signal magnitudes exceeding 5.3 volts.

Once the signal has been clamped, it is filtered by the 2-pole Butterworth, 20 kHz, anti-aliasing filter. The filter is implemented in a Sallen and key configuration, using the input operational amplifier (U103a) on the CS7008.

The CS7008 (U103b) is the programmable anti-aliasing filter for the A/D board. Both of the CS7008's power supply pins are decoupled by the parallel combination of a 1 uF tantalum electrolytic and a 0.1 uF monolithic capacitor. The signal output of the CS7008 is fed to the anti-image filter, composed of the uncommitted operational amplifier mounted on the CS7008 (U103c). The anti-image filter's characteristics are the same as the anti-aliasing filter used prior to the CS7008. The output of the anti-image filter is then amplified by an operational amplifier circuit (U102b) with a gain of 1.67. This amplifier makes the gain of the anti-aliasing filter network equal to 1 when the CS7008 is configured in an all-pass mode. The digital interface circuitry for the CS7008 will be presented later in this chapter.

<u>The signal-level comparator</u>

One of the required trigger sources for the analog-to-digital board was the signal level. The method of generating this trigger is shown in Schematic 4.3. The signal at the input of the sample-hold amplifier is compared with a voltage between -FS and +FS volts, where

FS is the full scale values (±5 volts). This comparison voltage is user-adjustable by a variable resistor (R120). This comparison is made with an LM311N comparator (U105). The comparator's output is used later in the trigger selection circuit. A hysteresis network (R122, C118) was installed between the comparator's output and input to prevent oscillations during transitions. The comparator's power supply pins are decoupled by 0.1 uF capacitors.

<u>The sample-and-hold amplifier and analog-to-digital converter</u>

The sample-hold amplifier (U105) is shown in Schematic 4.4. The SHA selected for this circuit was the AD346. The SHA is switched into the "hold" mode when "A/D_EOC" is not active (low). When power is first applied to the board, the ADC's EOC* signal may not become active until the board is reset. To prevent the output of the SHA from drifting to a power supply rail voltage while the board is waiting to be reset, a digital circuit (U150b, U152d) ensures that the SHA hold mode may be active only while the "board error" signal is not active ("board error" is active from the time power is first applied to the board until the board is initialized). The connection of the AD578K analog-to-digital converter (U106) is straight forward. The AD578K is connected in a manner to facilitate bipolar conversion, and by connecting the signal from the SHA to pin 27 (10V span), the full-scale input range is ±5.000 volts. Adjustments for bipolar offset are made available to the user by way of two multi-turn variable resistors (R124, R125). Details for making these adjustments are in Appendix A.

The conversion time for the AD578K is controlled by a resistor (R123) between the "clock adjustment" pin and the clock in/out pins. In order to take full advantage of the fast conversion rate of the K-version AD578, a 3.32 kohm 1% resistor was used as per instructions in [7].

All power supply leads on the AD578K are decoupled with a 6.8 uF tantalum capacitor in parallel with 0.1 uF monolithic capacitor. Additionally, the common connecting point for the digital and analog ground on the A/D board is at the analog ground pin on the AD578 (pin 30).

4.4.2. Timing Considerations and the Controlling Sequence for the Analog Circuit

The purpose of this section is to present the timing considerations made with respect to controlling the analog circuitry. Timing considerations are present in two forms: timing of the A/D converter during data conversion, and the timing associated with acquiring multiple samples.

The timing diagram in Fig. 4.4.2.1 illustrates the timing sequence for the AD578L A/D converter. At the

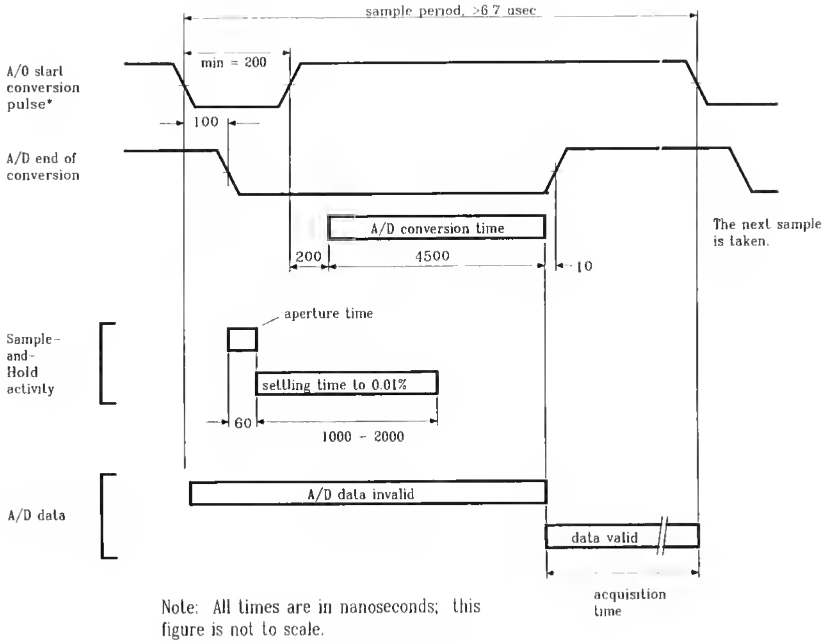


Figure 4.4.2.1 Timing diagram for single sample conversion.

receipt of a start-conversion pulse, the A/D begins its conversion. While the A/D is in a conversion mode, the "end of conversion" signal from the A/D causes the sample-hold amplifier (SHA) to "hold" the signal. Approximately 3 microseconds following the rising edge of the start-conversion pulse, the "end of conversion" signal becomes active which means the converted data may be retrieved from the A/D.

The controlling sequence from the digital controller's standpoint is divided into two sections. Fig. 4.4.2.2 illustrates the transition that occurs when there is a transition between "standby" mode to a conversion mode, at which time "standby" is no longer active. When the conversion mode is selected, two things occur within the circuit: (1) the on-board memory address generator resets to address 0000; (2) board-error, memory-cycle, and trigger-received status flags are reset to non-active.

In this example, the conversion mode is "convert immediately" which means the "acquisition enable" line

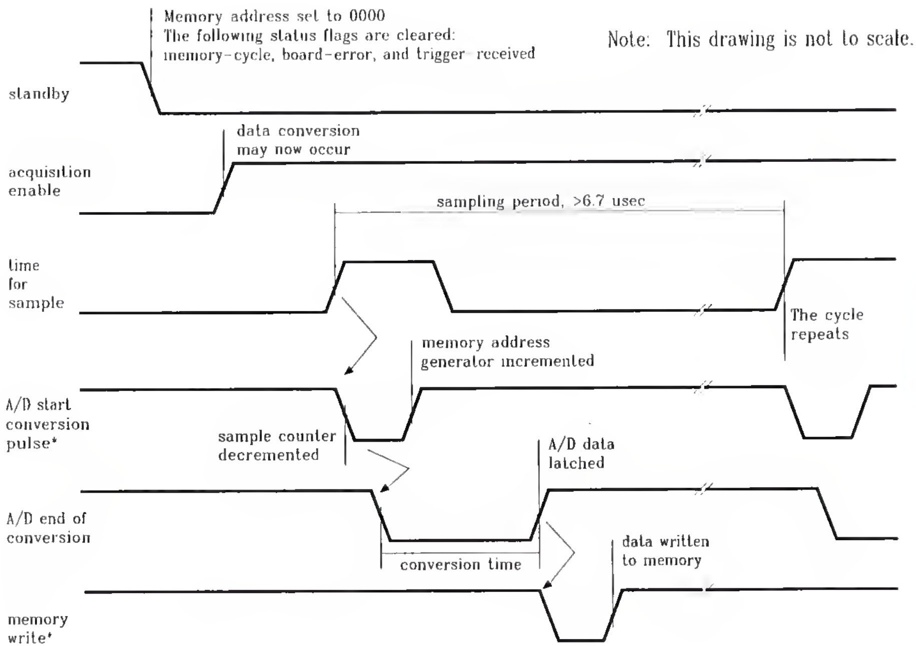


Figure 4.4.2.2 Timing diagram for beginning of acquisition sequence.

becomes active immediately.* As soon as the "time for sample" line becomes active, the A/D conversion pulse is generated, thus initiating the analog-to-digital conversion sequence shown in Fig. 4.4.2.1. Other things that occur at the receipt of the "time for sample" signal include: the sample counter is decremented; the memory address generator is incremented.

At the receipt of the rising edge of the "end of conversion" signal, the data is latched into the on-board memory at the address pointed to by the memory address generator. As soon as "time for sample" becomes active, this acquisition process begins anew until the desired number of samples has been acquired.

The end of the conversion sequence is indicated when the "end of sample sequence" line becomes active, as shown in Fig. 4.4.2.3. The "end of sample sequence" indicates that the desired number of samples has been acquired. The last sample acquired is much like all other samples, with the exception that the "end of sample sequence" becomes active as soon as the sample counter is decremented, thus inhibiting future sample requests.

---

* If the conversion mode was "convert at receipt of trigger", the "acquisition enable" line would become active at the receipt of the trigger thus enabling the conversion process.

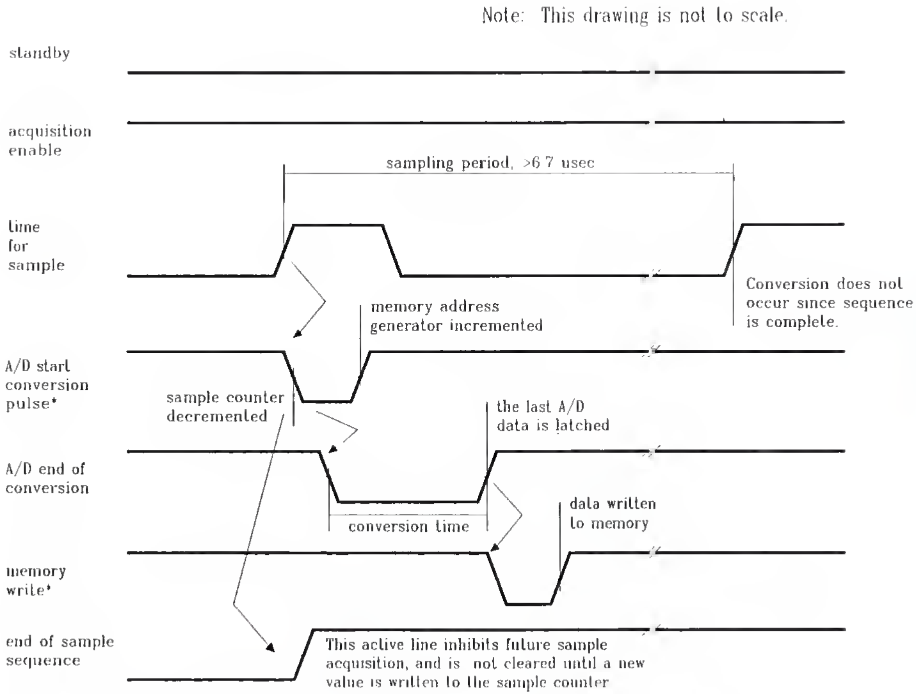


Figure 4.4.2.3 Timing diagram for the end of an acquisition sequence.

4.4.3. The Digital Controller Section

The purposes of the digital controller section of the A/D board include:

* control the data acquisition process (as detailed in section 4.4.2);
* retention of samples in on-board memory and making these samples available to the system controller;
* reporting the status of the acquisition process to the system controller.

The digital controller section is composed of several major sections, including the conversion control circuitry, the on-board sample memory, the control and status registers, as well as the system bus interface.

<u>Conversion control</u>

The circuit which commands the A/D converter to acquire samples at the appropriate time is the conversion control circuit, as was shown previously in Fig. 4.4.2.2 and 4.4.2.3. The conversion control circuit is composed of several sections, including the conversion rate selector, the trigger sensor and selector, and the sample counter. Each of these sections work together to establish the time when sampling is to begin, the period of the sampling interval, and the total number of samples in the acquisition sequence.

The conversion rate controller. The first important component of the conversion control circuit is the conversion rate controller. This circuit selects the sampling period for the conversion sequence. The configuration of this circuit is shown in Fig. 4.4.3.1.

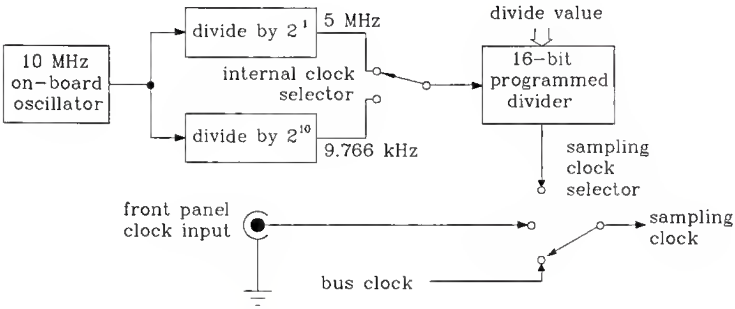


Figure 4.4.3.1 A block representation of the on-board sample period generator and the sample clock selector.

Fig. 4.4.3.1 illustrates that the conversion rate controller is made up of two sections: the sampling clock selector and the on-board sample rate generator.

The sampling clock selector provides a means to choose the sampling clock from three sources: 1) the bus clock, 2) the front-panel sampling clock input, and 3) the on-board sample rate generator. The bus clock is useful for synchronized acquisition with another board in the system (see chapters 2 and 3 for details about the bus clock). The front-panel clock is useful for making conversions at a rate determined by an external oscillator.

The other selectable clock source is the on-board sample rate generator. The specifications for the board state that the conversion rate must be programmable from 0.2 to 150,000 samples per second in reasonably fine steps. The method by which this sampling frequency is obtained is by a clock divider. The sampling frequency is thus determined by

$$F_{sample} = \frac{f_{osc}}{n} \text{ Hz}$$

where F_{sample} is the sampling frequency, F_{osc} is the frequency of the on-board oscillator, and n is the clock divider value. The counter chip selected for this circuit was the Intel 82C54-2, which has three programmable 16-bit counters. This is convenient since there were three counters used in the A/D board design. Unfortunately, 16-bits is not enough to cover the desired range of conversion rates. Therefore, two internal oscillator frequencies were made available: 5 MHz for fast sampling rates, and 9.776 kHz (10 MHz / 2^{10}) for slower sampling rates. Each of these frequencies may be divided by the

16-bit counter. The selection between the two internal oscillator frequencies depends on the desired sampling rate. The range of useful sampling frequencies for each internal oscillator frequency are presented in Table 4.4.3.1.

Table 4.4.3.1 The two on-board oscillators and their respective range of sampling frequencies.

Oscillator frequency	useful sampling frequency range minimum	maximum
9.77 kHz	0.15 Hz	100 Hz
5.0 MHz	100 Hz	150 kHz

The conversion rate controller circuit is shown in Schematic 4.5. The primary oscillator (from which the 5 MHz and 9.776 kHz frequencies are obtained) is composed of a 74HC04 inverter (U200) with a 10 MHz crystal (X200). The two frequencies are obtained by dividing the 10 MHz signal with a 74HC4040 clock divider (U201). The selection between the 5 MHz and the 9.776 kHz clock is made by a 4-to-1 multiplexer (U202a) where the clock selected by the multiplexer is used by the 16-bit binary counter (U203a). The control of this counter is described later. Finally, a 4-to-1 multiplexer (U202b) is used to select the sampling clock from the internal-clock, the bus clock, and the front-panel clock.

The connection for the front-panel clock is also shown in Schematic 4.5. The external clock is connected to a BNC-connector (J200). A clamping network (R201, D200, D201) ensures that the clock signal provided to the multiplexer (U202b) is TTL-compatible.

Trigger selector and identifier circuit. Fig. 4.4.3.2 illustrates the duties of the trigger circuit on the A/D board.As shown, one of three different trigger sources may

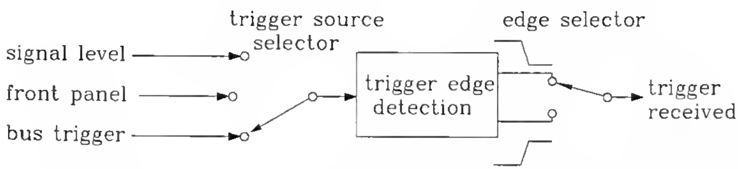


Figure 4.4.3.2 Block representation of the A/D board's trigger circuit.

be selected: signal-level trigger, the front-panel trigger connection, and the bus trigger. The source selected by the trigger source selector is then provided to the trigger edge detection circuit. This circuit identifies either a rising or falling edge.

The trigger circuit used on the A/D board is shown in Schematic 4.6. The three trigger sources are input to a 4-to-1 multiplexer (U204) where the source is selected by two control register lines (trig_sel_0, trig_sel_1). One of these trigger sources is the external trigger which is connected to the front-panel BNC trigger connector (J201). The external trigger is clamped (R203, D202, D203) to protect the A/D board circuitry from non-TTL level signals.

The trigger edge of the selected trigger source is detected by an edge-sensitive flip-flop (U205a). An exclusive-OR gate (U252c) is used to make the flip-flop rising or falling edge sensitive. At the beginning of an acquisition sequence, the flip-flop is cleared by a pulse (rst_before_acq*) and its outputs (trig_recvd*, trig_recvd) becomes active when the selected edge is detected.

The last part of the trigger circuit provides a means for the A/D board to activate the bus trigger. When the trigger selection is "signal with bus control" and a trigger from the signal is sensed, an open-drain gate (U262b) pulls the bus-trigger line low, thus triggering other boards monitoring the bus trigger.

<u>The sample counter</u>. Once the conversion sequence begins, the system must stop acquiring samples when the desired number of samples has been acquired. Therefore, a 16-bit binary counter was used to count post-trigger samples. Note that not all conversions are counted. Two exceptions are: 1) conversions made prior to a trigger reception (in

the pre-trigger acquisition mode), and 2) acquisitions requested when the board is in the STANDBY mode (this results from a "write" operation to register 7). The truth table for the sample counter controller is shown in Table 4.4.3.2.

Table 4.4.3.2 Truth table for sample counter controller.

Conversion mode	Trigger received?	Sample counter clock enabled?
Standby	X	NO
convert immediately	X	YES
conv. w/trigger modes	NO	NO
conv. w/trigger modes	YES	YES

X = don't care

Schematic 4.7 shows the sample counter circuit and the count-enable circuit. The 16-bit binary counter used is the second of three counters on the 82C54-2 (U203b). The count-enable circuit (an implementation of the truth table in Table 4.4.3.2) provides clock pulses to the sample counter. When the number of pulses received by the counter equals the number of samples requested, the sample counter indicates the sequence is complete (samp_ser_end becomes active).

The number of desired samples plus one (N+1) is converted to a 16-bit binary number and loaded into the counter via register 2. After N pulses have been received, the output of the counter becomes active. Once active, the conversion sequence ceases and the "acquisition sequence completed" status bit becomes active.

<u>Conversion control logic</u>. As shown in Fig. 4.4.3.3, the signals generated by the sampling clock, the trigger sensor, and the sample counter are all used by the conversion logic unit. The conversion logic unit

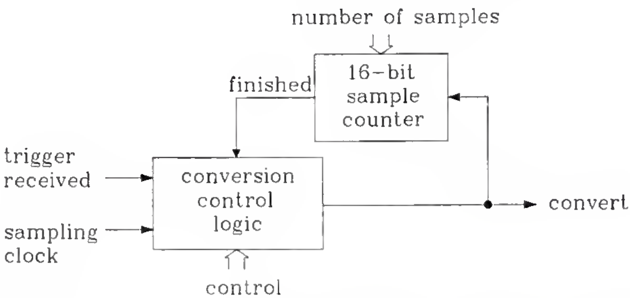


Figure 4.4.3.3 A block representation of the conversion control logic circuit.

implements the truth table given in Table 4.4.3.3.

The conversion control circuit shown in Schematic 4.8 fulfills the logic requirements depicted in Table 4.4.3.3. When it's time for a sample (time_for_sample* active), a rising edge is present at the gate of the digital one shot (U203c). This digital one-shot then generates the conversion request pulse (a/d_conv*), where the duration

Table 4.4.3.3 The conversion control logic truth table.

conversion mode	trigger received?	single sample requested?	end of sample sequence?	board error?	conversion request
X	X	NO	NO	NO	NO
X	X	YES	X	X	YES
X	X	NO	X	YES	NO
X	X	NO	YES	X	NO
immediate	X	NO	NO	NO	TFS
on trigger	NO	NO	NO	NO	NO
on trigger	YES	NO	NO	NO	TFS
pre-trigger	X	NO	NO	NO	TFS

X = don't care
TFS = time_for_sample signal (from sampling clock generator)

of this pulse is 800 ns (as outlined in section 4.4.2: analog circuit control timing).

On-board sample memory

One of the requirements for the A/D board is that it has its own on-board memory. This on-board memory provides temporary storage of conversion values until the system front-end can retrieve them and send them to the host. Fig. 4.4.3.4 shows a simplified block diagram of on-board memory and its associated control circuitry.

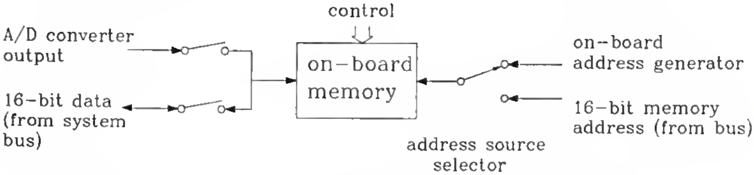


Figure 4.4.3.4 A block representation of the A/D board's on-board memory.

The data source for the on-board memory is selected by two source selection switches. When the A/D board is in a conversion mode, the data source is the A/D converter, and when the A/D board is in the "standby" mode, the data source is the system data bus. The arrangement of the data selection switches at the input of the on-board memory makes it possible to put the A/D data directly on the system data bus.

The address source for the on-board memory, like the data source, is dependent upon the board's operating mode. While the board is in a "conversion" mode, the address source is the internal address generator. When the board is in the "standby" mode, the address source is the memory address lines from the system bus.

As shown in Fig. 4.4.3.5, the on-board address generator has two control lines: clear and increment.

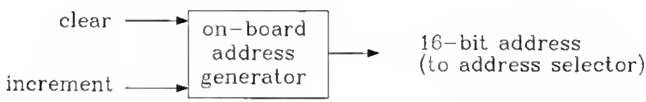


Figure 4.4.3.5 A block representation of the on-board address generator.

Before a sampling sequence begins, the address generator is cleared, and as samples are stored in memory, the address generator is incremented. Since the maximum number of samples to be retained in the on-board memory is 64K, the address generator is simply a 16-bit binary counter.

Schematic 4.9 shows the circuit used for the on-board memory control. A one-shot implemented in a pair of flip-flops (U207) generates a 400 ns write pulse when the A/D converter completes a conversion (a/d_eoc active). An "OR" gate (U251c) permits the source of the memory write request to originate from the pulse generator or a write to register 1 (on-board memory register). An "AND" gate (U264a) disallows writes to on-board memory from the bus while the A/D board is in a non-standby mode.
Two additional gates (U257c, U263b) provide data buffer control.

The circuit shown in Schematic 4.10 is the memory data switches. Two octal latches (U208, U209) capture the data from the A/D converter when the conversion is completed (a/d_eoc becomes active). Pull up resistors are installed on the inputs of these latches to protect the latches in the event the circuit is powered up while the A/D converter is removed. A pair of octal transceivers (U210, U211) constitute the data switch between the data bus and the on-board memory.

As shown in Schematic 4.11, the on-board address generator is composed of two cascaded 8-bit binary counters (U212, U213). The 16-bit address is reset to 0000 prior to a conversion sequence (rst_before_acq pulse), and the memory address is incremented when a conversion is requested (a/d_conv* pulse). A flip flop circuit provides the status concerning whether or not memory has cycled during a conversion sequence. The flip flop (U214a) is cleared prior to the acquisition sequence (rst_before_acq* pulse) and becomes active on the falling edge of the most significant address bit (when the address goes from $FFFF to $0000).

The memory address selector for the on-board memory is made up of four octal transparent latches. While the A/D board is in a "conversion" mode, the latches shown in Schematic 4.12 (U215, U216) are enabled, thus making the address source the on-board address generator. When the A/D board is in "standby" mode, the 16-bit memory address lines from the system bus provide the on-board memory's address. This address source is connected by latches shown in Schematic 4.12 (U217, U218).

Schematic 4.12 also has a pair of clocked 8-bit latches (U219, U220). These latches record the memory address when the trigger is received (for pre-trigger sample location calculations), where upon this 16-bit address is made available to the data bus during a "read register 5" operation.

The memory elements selected for the A/D board were four 43256, 32Kx8 bit static RAM devices (U221-U224) since they were readily available during the board's design. This configuration of memory devices provides a memory space of 64Kx16 bits. Schematic 4.13 shows the memory components responsible for the lower half of 64K sample memory (addresses $0000 - $7FFF), and Schematic 4.14 shows the memory components for the upper half (addresses $8000 - $FFFF).

Control and status registers

There are numerous control and status lines on the A/D board. A convenient way to accommodate these signals was to use an 82C55A triple 8-bit parallel peripheral interface (U225), as shown in Schematic 4.15. The 82C55A permits two 8-bit registers to be configured as an "output" (control) and the remaining 8-bit register configured as an "input" (status). Registers on the 82C55A that are configured as "outputs" (the control lines) can be read, thus enabling the settings of the control lines to be examined by reading that control register.

When power is first applied to the board, POWER_ON_RESET causes all I/O lines on the 82C55A to be configured as "inputs." 10 kohm pull-up resistors are connected to the control output lines (RN209 and RN210) to pull the control lines HIGH until the 82C55A is properly configured. Configuration of the 82C55A is made by writing a control code byte ($92) to register 11 (usually as part of the A/D board initialization routine).

Bus related I/O with the register on the 82C55A are performed by register 3 and 4 read/write operations in conjunction with the register address lines RA1 and RA3, resulting in the mapping of the 82C55A ports as shown in Table 4.4.3.4.

Table 4.4.3.4 A/D board register mapping of the control/status port.

A/D board register (decimal)	82C55A port description
3	status register
4	control register #1
11	82C55A configuration reg.
12	control register #2

Additional control related signals are generated by the circuit shown in Schematic 4.16. The conversion mode and signal selection lines from the control registers are each decoded by a 74HC139 dual 1-of-4 decoder (U226a and

U226b).

Relay control circuits

The relay control circuits are, in general, controlled directly by the control registers. These relays include those for input isolation, signal routing, and gain control. The power requirements for a relay solenoid is 10 mA, thus permitting the relay to be switched directly by a 74HC03 open-drain NAND gate.

The schematic for the isolation relay control circuit is shown in Schematic 4.17. A 2-input NOR gate (U152b) ensures that the isolation relays are not activated while a board error condition is present. When the "external signal" is the selected signal source, the shorting relay is opened, and, after a short delay, the input isolation relays (REL100, REL101) are closed. A 15 millisecond delay is inserted between the opening and closing of the relays to prevent the signal source from being shorted. This delay is generated by a dual one-shot (U107).

Schematic 4.18 shows the circuit used to implement the signal routing logic given earlier in Table 4.4.1.1. The circuit which controls the gain selection relays is shown in Schematic 4.19. The gain control circuit has two jumpers which ease instrumentation amplifier offset error adjustment. JMP102, when installed, disables the gain selection decoder, thus forcing the gain to 1.0. This is useful for OUTPUT offset error adjustments. JMP103, when installed, turns the "gain = 500" relay "on". This is useful for INPUT offset error adjustments. In both cases, the jumper shown in schematic ISO_IN , JMP101, must be installed to ensure the instrumentation amplifier inputs are shorted. The procedure for adjusting the I.A. offset is given in Appendix A.

<u>The error status generator and reset pulse generators</u>

A power-up reset pulse is made available to the A/D board by way of the circuit shown in Schematic 4.20. An RC-circuit in conjunction with a pair of Schmitt-trigger inverters provides a sufficiently long reset pulse when power is freshly applied to the board. Also shown in Schematic 4.20 is the "board-error status" generator. A board-error status can arise from two conditions: (1) power has been freshly applied to the circuit, or (2) the signal has overloaded the input amplifier. The board-error status signal originates from a flip-flop (U214b) where the outputs become active at the receipt of an error (U263a has an active output). The board-error status is reset when the board's conversion mode transfers from STANDBY to a conversion mode (at which time the rst_before_acq* signal pulses).

The outputs of the signal overload comparator (U101, Schematic 4.1) is buffered by a 20 milliseocnd delay circuit (U150c,d and R128, C130). This delay circuit prevents false triggering of the board-error circuit by requiring the duration of the overload to be at least 20 milliseconds before board-error is activated. An OR-gate (U264b) enables or disables the overload protection function; the overload protection circuit is enabled when protect* is active (LOW).

The circuit which generates the rst_before_acq pulse is also shown in Schematic 4.20. When the A/D board is switched to a conversion mode, "standby" becomes inactive, resulting in a 400 nanosecond pulse generated by a single-pulse generator (U227, U254a).

<u>The LED control circuit</u>

The LED driver circuits and associated logic are shown in Schematics 4.21a and 4.21b. Transistors (Q201-Q209) were used to switch the LED power since the LEDs require 25 mA for full illumination.

<u>The 82C54-2 (U203) digital interface</u>

Schematic 4.22 illustrates the data bus connections for the 82C54-2 triple counter (U203). The 8-bit counter I/O is by way of register 2 on data lines D8-D15. The 82C54-2 has four internal registers, where the register

being addressed is selected TIMER_CONT_0 and TIMER_CONT_1. The internal registers of the counter device are summarized in Table 4.4.3.5.

Table 4.4.3.5 The 82C54-2 internal register summary and associated control values.

TIMER_CONT_n 1	0	82C54-2 internal register description
0	0	counter #0: sample-period generator
0	1	counter #1: digital one-shot
1	0	counter #2: sample counter
1	1	82C54-2 config. register

<u>Digital considerations for the CS7008 switched-capacitor filter</u>

The characteristics of the CS7008 anti-aliasing filter (U103) used in the analog circuit are determined by coefficients written to the filter's configuration memory. Therefore, the CS7008's memory is accessible from the system bus via I/O operations with register 6.

As shown in Schematic 4.23, data for the CS7008 is present on D8-D13, or the six least-significant-bits of the most-significant-byte of data. The filter's address lines are present on A0-A6. The filter's chip select line becomes active whenever a read or write request is made to the filter. The value of the R/W* line is held steady by a 74HC75 bistable transparent latch (U112) while the chip select is active; this prevents data contention during read/write transitions. Protection from a loss of the CS7008's address and data signals (for whatever reason) is provided by 10 kΩ pull-up resistors (RN101, RN102).

<u>Selection of the CS7008 crystal</u>

The sampling frequency for the CS7008 is given by

$$f_s = \frac{f_{osc}}{6 * 2^{cdc}} \qquad \text{(Hz)}$$

where f_s is the CS7008's sampling frequency, f_{osc} is the crystal (X100) frequency (in Hz), and cdc is the clock division code stored, filter coefficient address $1E, where cdc is 0,1,...,7. Table 4.4.3.6 shows the sampling frequencies attainable using crystals from 1 to 4 MHz. Crystal specifies that f_s < 250 kHz, and f_{osc} < 4.0 MHz. Therefore, inspection of Table 4.4.3.6 reveals that a crystal frequency of between 1.0 MHz and 1.5 MHz provides the most flexibility with respect to setting the sampling frequency. However, a low-profile crystal (body style HC18) was not readily available within this range. Therefore, a 2.4576 MHz crystal was used. Table 4.4.3.7 presents the sampling frequencies which are available on the prototype.

Table 4.4.3.6 Comparison of sampling frequency, f_s, with respect to various crystal frequencies.

	f_{osc} (External XTAL frequency) in MHz						
cdc*	1.0	1.5	2.0	2.5	3.0	3.5	4.0
0	166.7	250.0	333.3	416.7	500.0	583.3	666.7
1	83.3	125.0	166.7	208.3	250.0	291.7	333.3
2	41.7	62.5	83.3	104.2	125.0	145.8	166.7
3	20.8	31.2	41.7	52.1	62.5	72.9	83.3
4	10.4	15.6	20.8	26.0	31.2	36.5	41.7
5	5.2	7.8	10.4	13.0	15.6	18.2	20.8
6	2.6	3.9	5.2	6.5	7.8	9.1	10.4
7	1.3	2.0	2.6	3.3	3.9	4.6	5.2

* clock division code (address $1E of the filter register).

Note: All frequencies in this table are in kHz.

Table 4.4.3.7 Available sampling frequencies, f_s, for the CS7008 with the 2.4576 MHz crystal installed on the prototype.

cdc*	sampling frequency, f_s (kHz)
0	409.6
1	204.8
2	102.4
3	51.2
4	25.6
5	12.8
6	6.4
7	3.2

* clock division code (address $1E of the filter register).

One obvious consequence of using the 2.4576 MHz crystal is that a zero cdc-value is illegal since f_s > 250 kHz, though the next step down, cdc = 1, permits a sampling frequency over 200 kHz.

Bus interface

The A/D board's bus interface consists of the following components: the board address decoder, data and address buffers, and the board's command interpretation EPROM. Details concerning the system bus and the generic I/O board interface are given in Chapter 3.

Board address decode. The board address lines on the bus (BD_ADR0-3) select the board with whom bus communication is requested. As shown in Schematic 4.24, the board address comparator circuit consists of a 74HC85 4-bit magnitude comparator (U228) and a four station DIP switch (SW200) by which the board address is selected. Upon favorable equality comparison BD_SELECT* becomes active.

Register read/write control. Schematic 4.25 shows the circuit used for register address decoding. The register address lines on the bus (REG_ADR0-3), and the bus write and read strobes (BUS_WR* and BUS_RD) are buffered by a 74HC573 tri-stateable transparent buffer (U229). This buffer is transparent ONLY when the board is being addressed, and tri-stated during all other conditions (thus locking out all register I/O with the A/D board when it is not selected). These signals are tri-stated when the board is not selected to reduce digital activity on the A/D board when another board on the bus is being addressed. While the buffer is tri-stated, each of the buffered lines is pulled high by 10 kohm pull-up resistors.

When the board is selected, these buffered signals are provided to a pair of 74HC138 3-to-8 decoders: one for "write" (U230) and "read" (U231). An exclusive-OR gate (U252b) ensures that both decoders are disabled until READ* and WRITE* are opposite, as is the case during a bus read or bus write, at which time the appropriate decoder is enabled.

Two other signals are generated by the register read/write control circuit which control the data buffers. The data direction of the bus data (D0-D15), as determined by the action of the BUS_RD* signal, is provided to the data bus buffers as DATA_DIR_RD*. Additionally, an enable line, DATA_BUF_EN*, is provided to the data buffers. This signal is active during valid bus read/write activity (determined by U252b).

<u>16-bit bus data buffer</u>. Schematic 4.26 shows the 16-bit bus data buffers. Data to and from the A/D board is

buffered by a pair of 74HC245 octal transparent tri-stateable transceivers (U232 and U233). The data direction is controlled by DATA_DIR_RD*, and the buffer outputs are enabled by DATA_BUF_EN*. To avoid floating lines between read/write operations, the A/D board data lines (D0-D15) are pulled high by 10 kohm pull-up resistors.

16-bit memory address buffers. 16-bit memory address signals (BA0-BA15) present on the bus are also used by the A/D board. Schematic 4.27 to a pair of 74HC573 octal transparent buffers with tri-stateable outputs (U234 and U235). The address lines on the A/D board side (A0-A15) are copies of BA0-BA15 when the board is selected (BD_SELECT* is active), and pulled high by 10 kohm resistors otherwise. Tri-stating the address line while the board is not being addressed reduces digital activity on the A/D board while another board is being addressed.

EPROM and board presence . As mentioned in Chapter 3, all I/O boards must have an EPROM containing command implementation programs for the board's commands. The A/D board is no exception. As shown in Schematic 4.28, an 8Kx8 EPROM (U236) is located on the A/D board. As shown, the address lines A0-A12 selects the memory position, and the 8-bit data from the EPROM is placed on data lines D0-D7 during a read from register 0.

Board presence is indicated when a read from register 0 is requested and the most significant bit (D15) is pulled LOW by an open-drain NAND gate (U262c).

A/D board power supplies

All power for the A/D board is supplied by the system bus. The +5 volt power for the digital components has all ready been regulated at the system front end, and therefore needs only to have a moderate sized filter capacitor on the A/D board. The analog signal components need ±5.0 and ±15.0 volts, and this is regulated from the ±19 volts available on the bus.

Schematic 4.29 shows the power as it is taken from the system bus. The ±19 volts for the analog components is connected directly to a connector (CON203) which supplies the analog board. The +5 volt supply is filtered by a 47 uF electrolytic capacitor (C204), and the power supply pins on each of the digital logic chips aboard the A/D digital board are decoupled by way of a 0.1 uF monolithic capacitor (C205-C252).

The power needs for the analog board satisfied by the circuit shown in Schematic 4.30. Power from the digital board connector (CON203) is connected to a connector on the analog component board (CON103). An LM325 tracking regulator (U113) provides the ±15 volt supply, and an LM341/LM320 pair provide the ±5 volt supply for the analog signal components. The +5 volts for the digital components is first filtered by a 6.8 uF tantalum (C134), and all power supply pins of the analog board logic devices are decoupled by 0.1 uF monolithic capacitors (C135-C143).

4.4.4. A/D Board Circuit Schematics and Parts List

The following pages are the schematics for the A/D board. Each schematic was described in an earlier section of this chapter. A list of the parts used in these circuits follows the schematics. Part numbering was determined by the physical position of the component, where components numbered from 100-199 are located on the analog signal board, and components numbered 200-299 are located on the digital control board. Further information concerning the placement of the components is presented in Appendix D.

The following notes apply to the schematics:

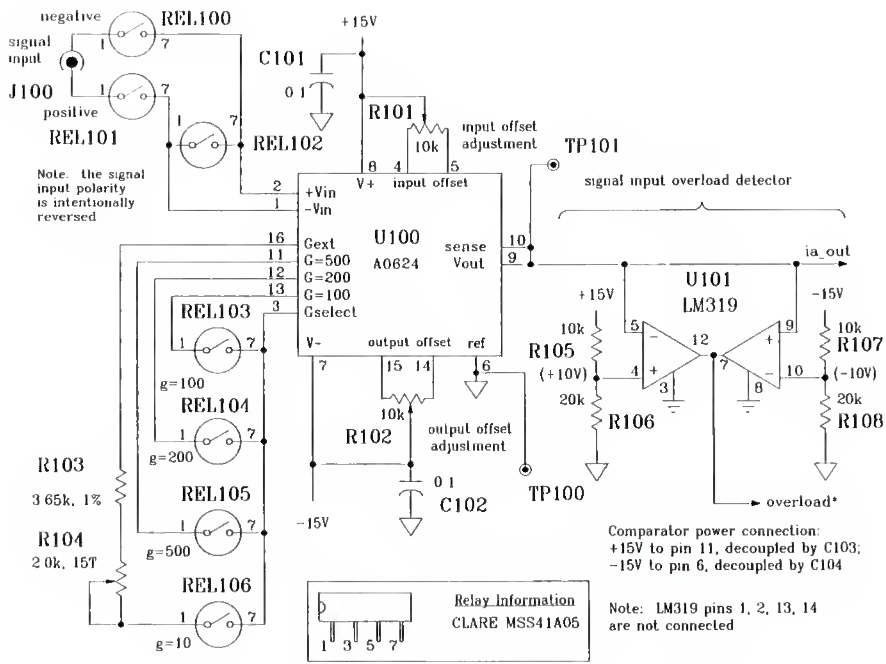
1) Ground symbols. The A/D board employs two distinct grounds, an analog and digital ground. To distinguish these grounds in the schematics, the digital ground is represented with the traditional 3 parallel lines, and the analog ground is represented by a triangle.

2) +5V power. Two distinct +5-volt power sources are used on the A/D board: the +5V used by the analog signal components (supplied by U114), and the +5V used for the digital components (supplied directly from the bus). These two sources are distinguished by the label (analog) for the +5V analog power source. A +5V reference, when unaccompanied by the before mentioned label, implys that the supply is intended for the digital components.

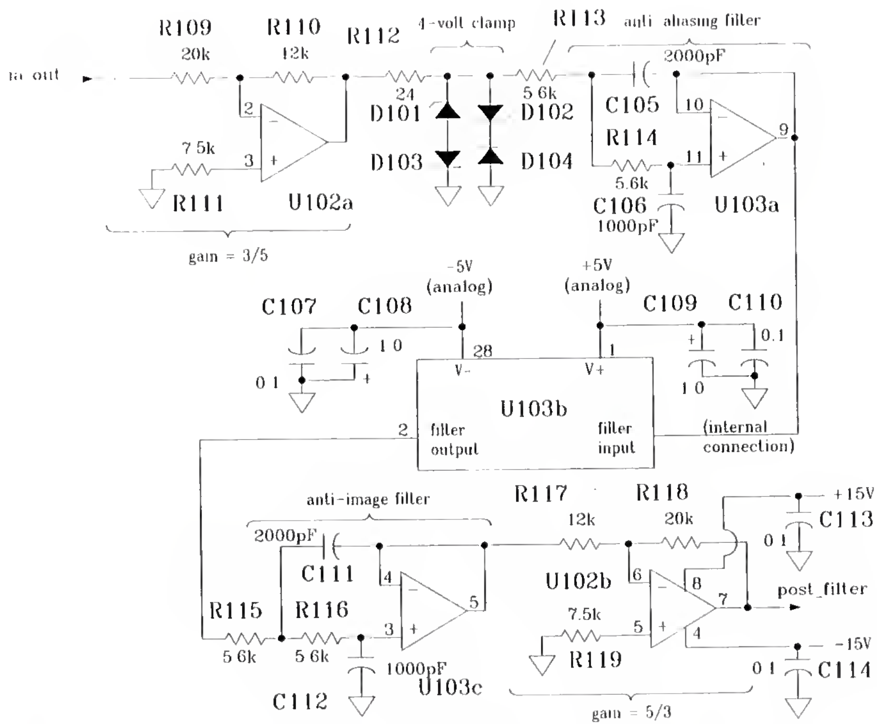
3) All signal names in the schematics with a "*" suffix indicate that the signal is "negative-true" e.g. a LOW logic level = TRUE.

4) All wire connections in these schematics are shown with a "solid dot". Wires which cross without a solid dot are not connected.

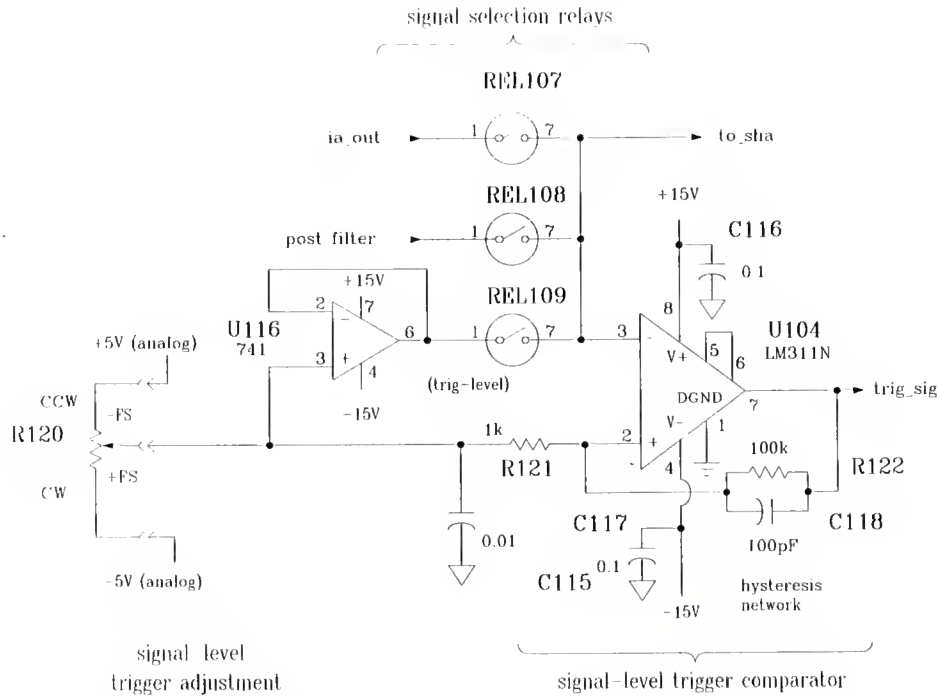
An "inter-figure signal" description list follows the schematics. This alphabetical list is useful when tracing signals from one schematic to another: it provides both a description of the signal and information concerning the schematic on which it originates and the schematics on which it is used.



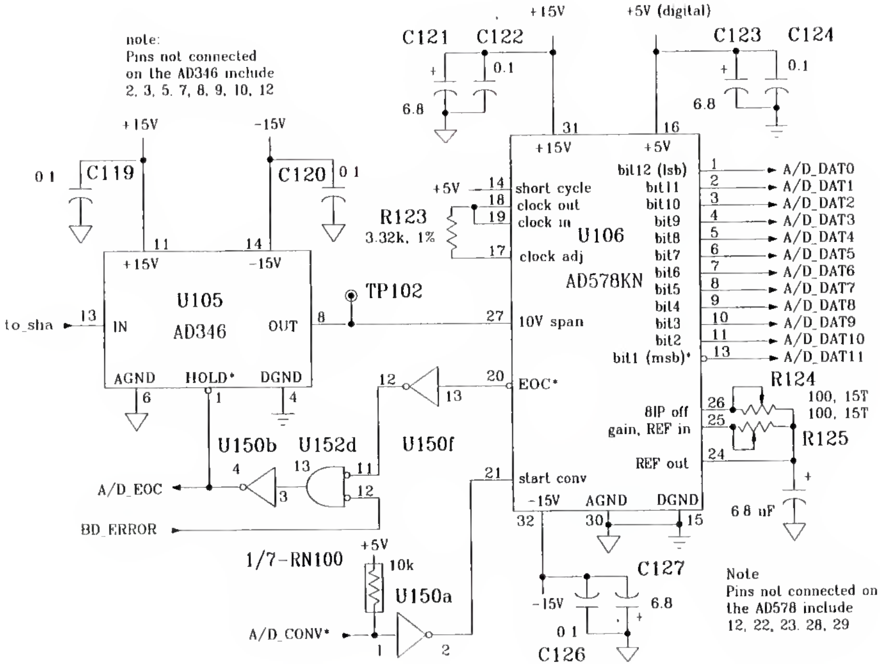
Schematic 4.1 The instrumentation amplifier and the overload detector.



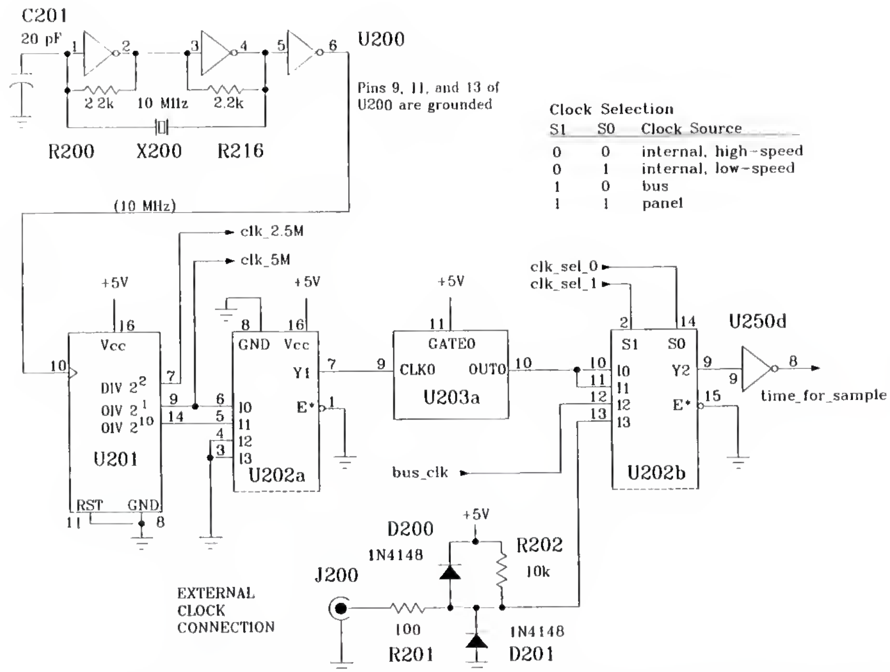
Schematic 4.2 The on-board anti-aliasing filter and its associated circuitry.



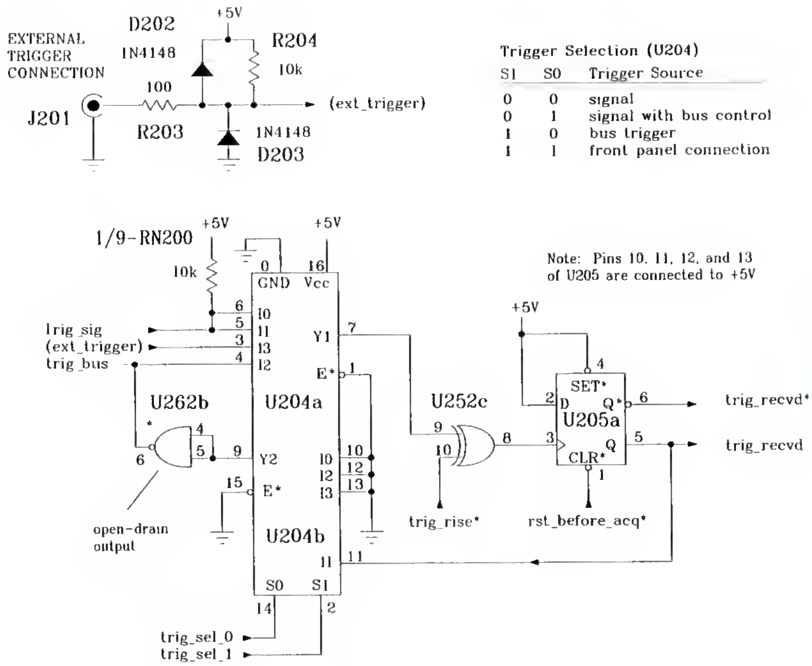
Schematic 4.3 The signal selection relays and the signal level trigger detector.



Schematic 4.4 The sample-and-hold amplifier and the analog-to-digital converter.



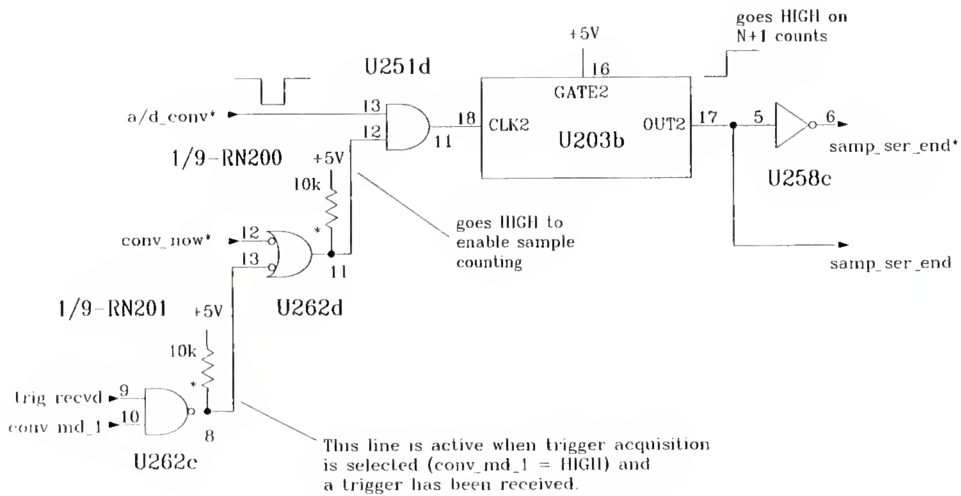
Schematic 4.5 The on-board sampling period generator, and the sampling clock selector.



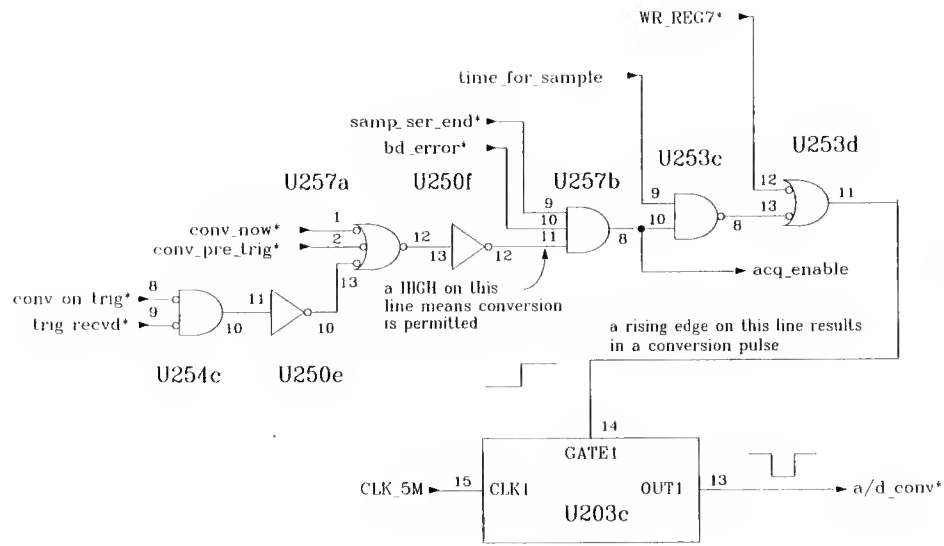
Trigger Selection (U204)

S1	S0	Trigger Source
0	0	signal
0	1	signal with bus control
1	0	bus trigger
1	1	front panel connection

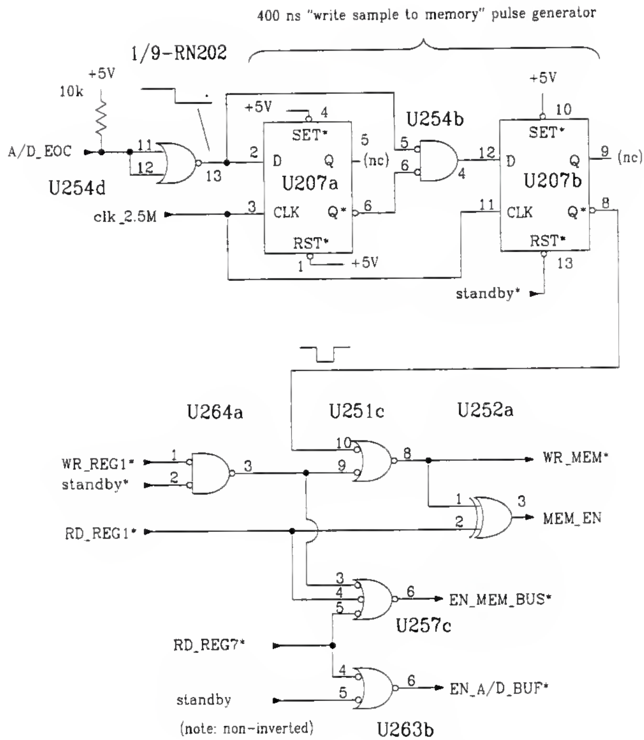
Schematic 4.6 The trigger source selector and trigger detection circuit.



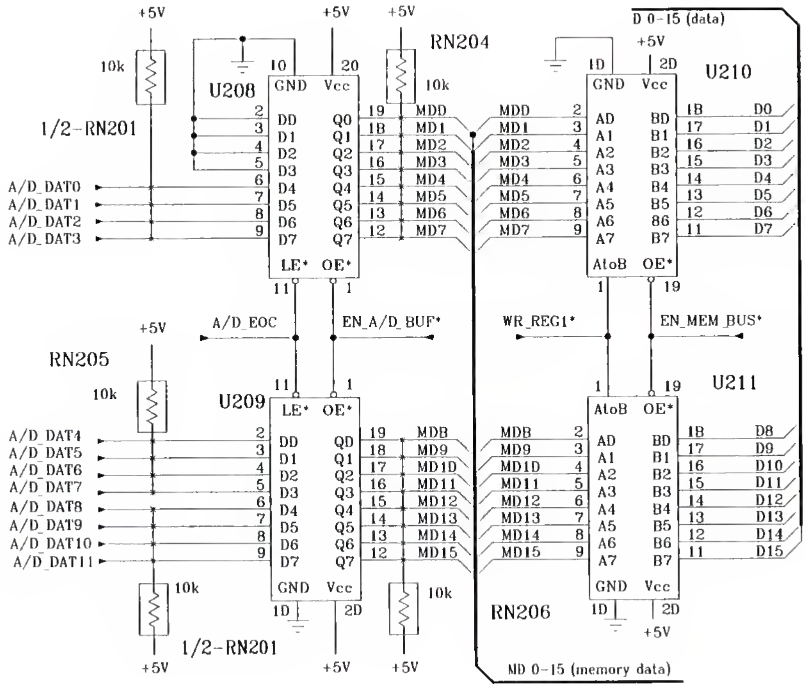
Schematic 4.7 The sample counter for the conversion controller.



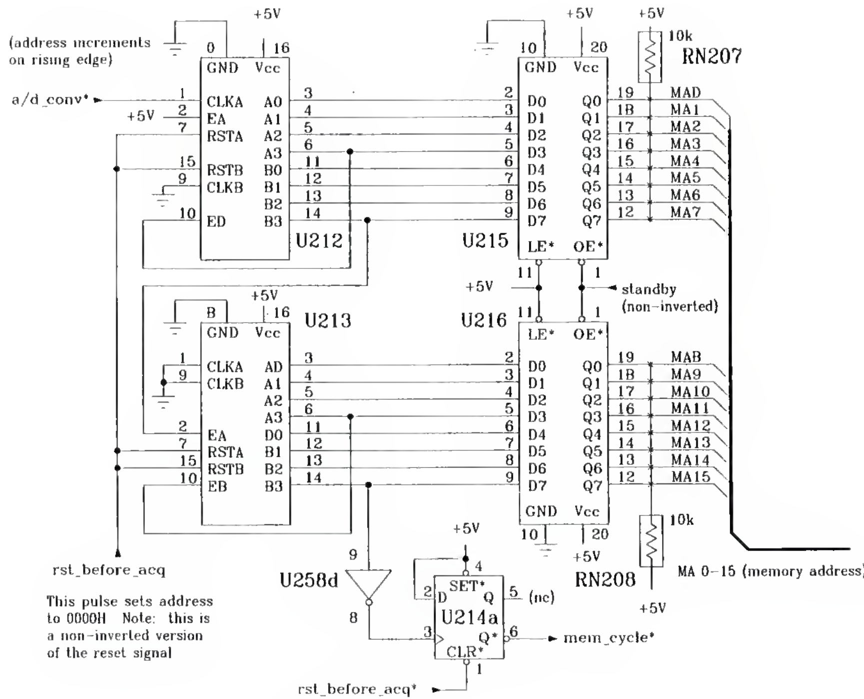
Schematic 4.8 The convert-enable circuit and the conversion pulse generator.



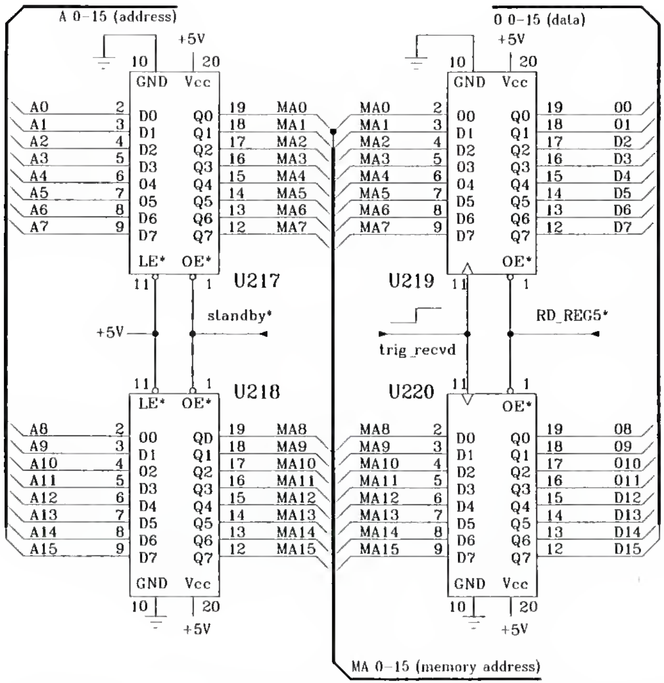
Schematic 4.9 The on-board memory control, including the data buffer controllers and write-pulse generator.



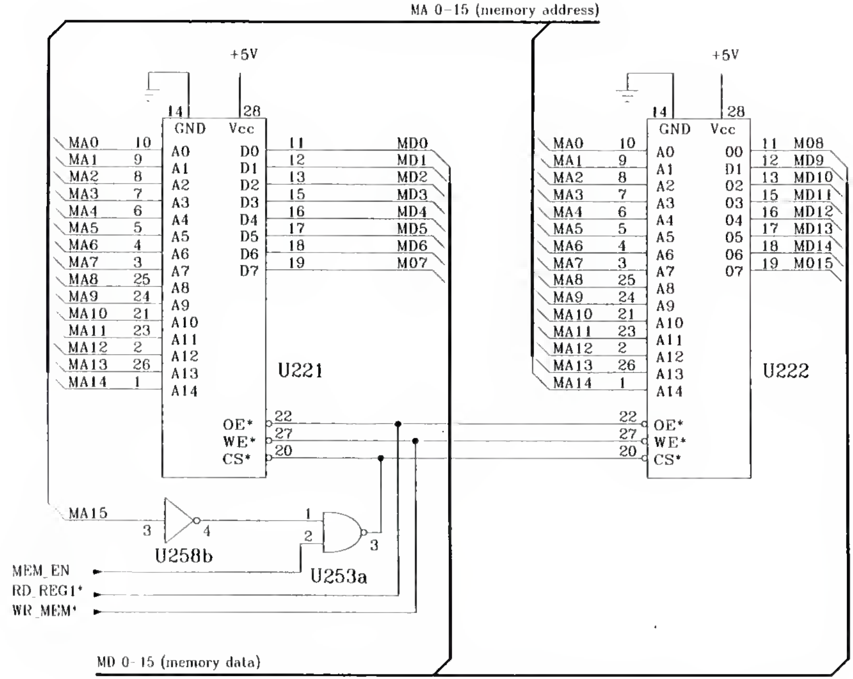
Schematic 4.10 The on-board memory data source selection circuit.



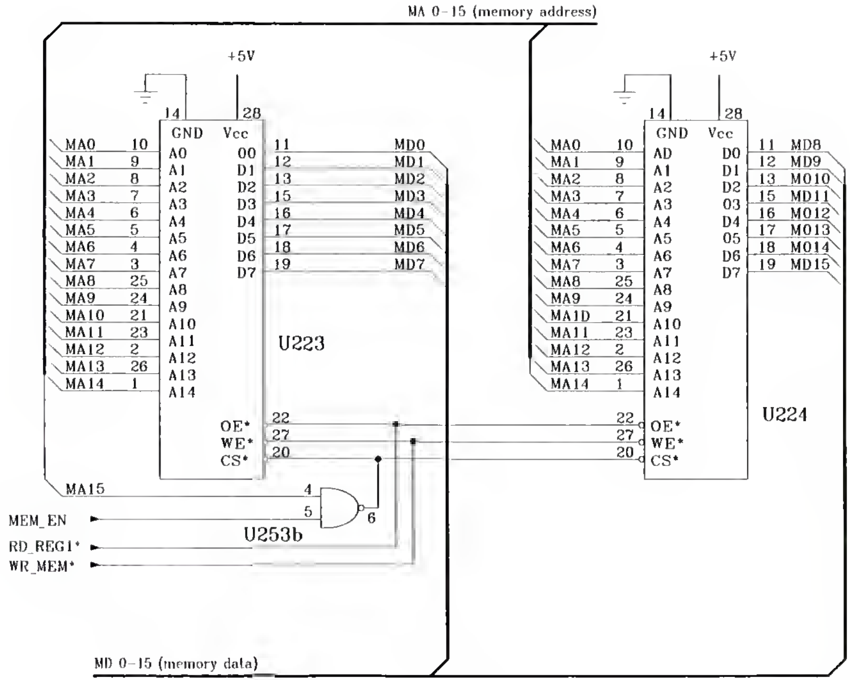
Schematic 4.11 The on-board memory address generator circuit.



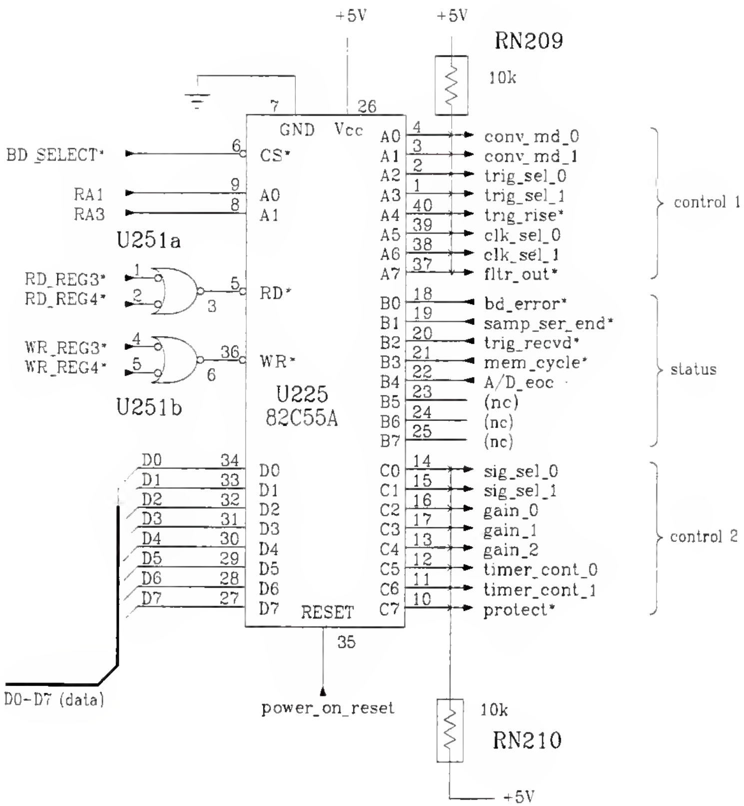
Schematic 4.12 The on-board memory address source selector and trigger-address capture latches.



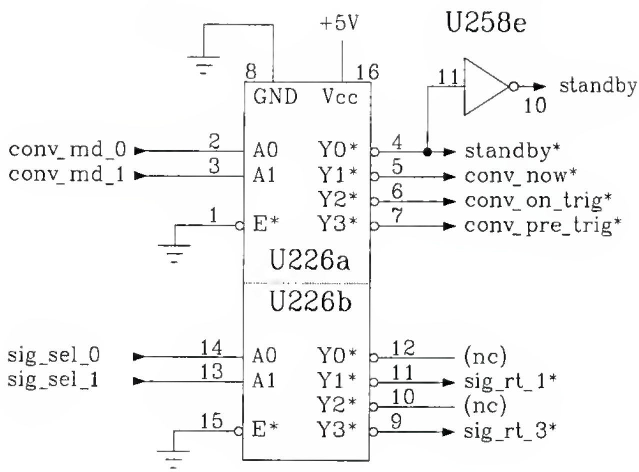
Schematic 4.13 The on-board memory devices: addresses $0000 - $7FFF.



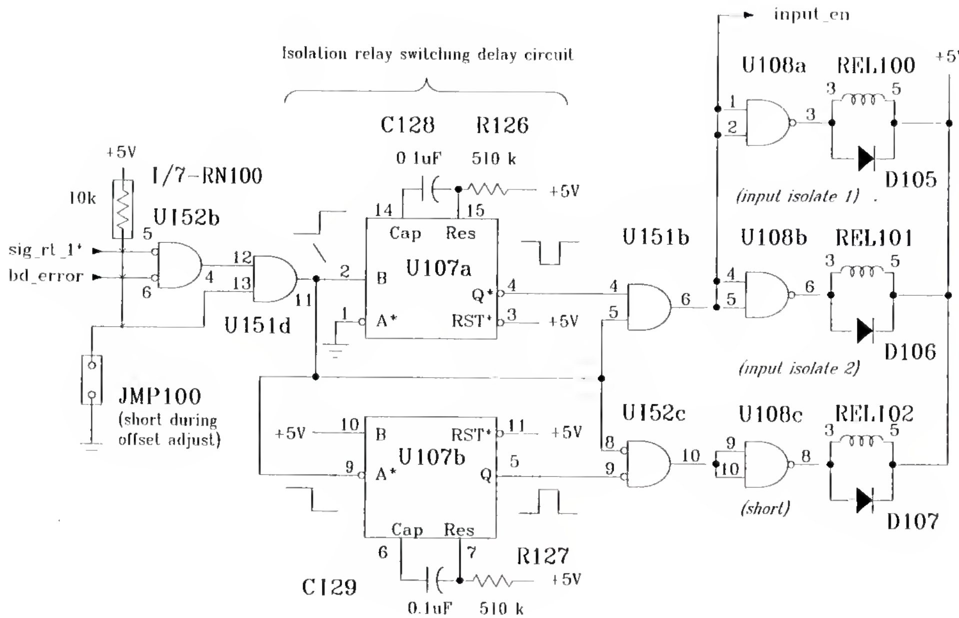
Schematic 4.14 The on-board memory devices: addresses $8000 - $FFFF.



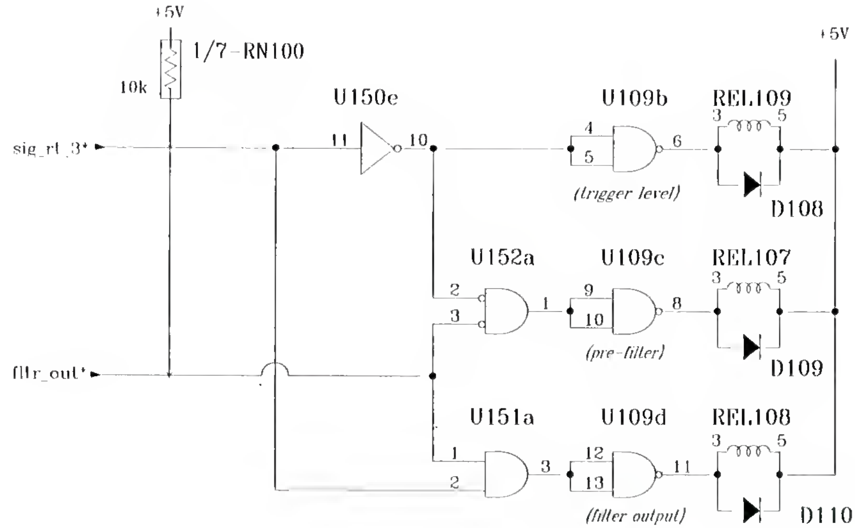
Schematic 4.15 The A/D board status and control registers.



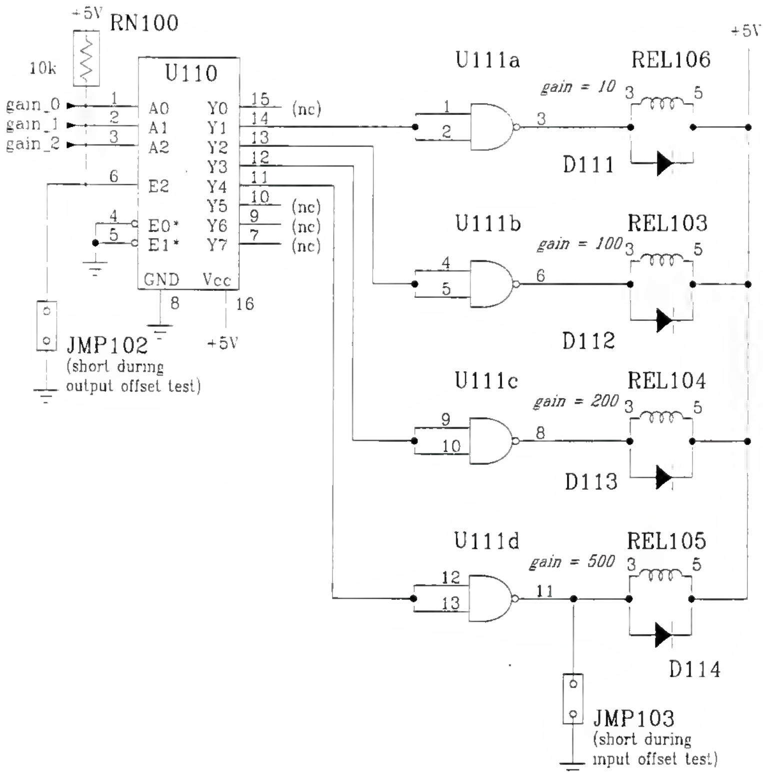
Schematic 4.16 The conversion mode decoder and the signal route decoder.



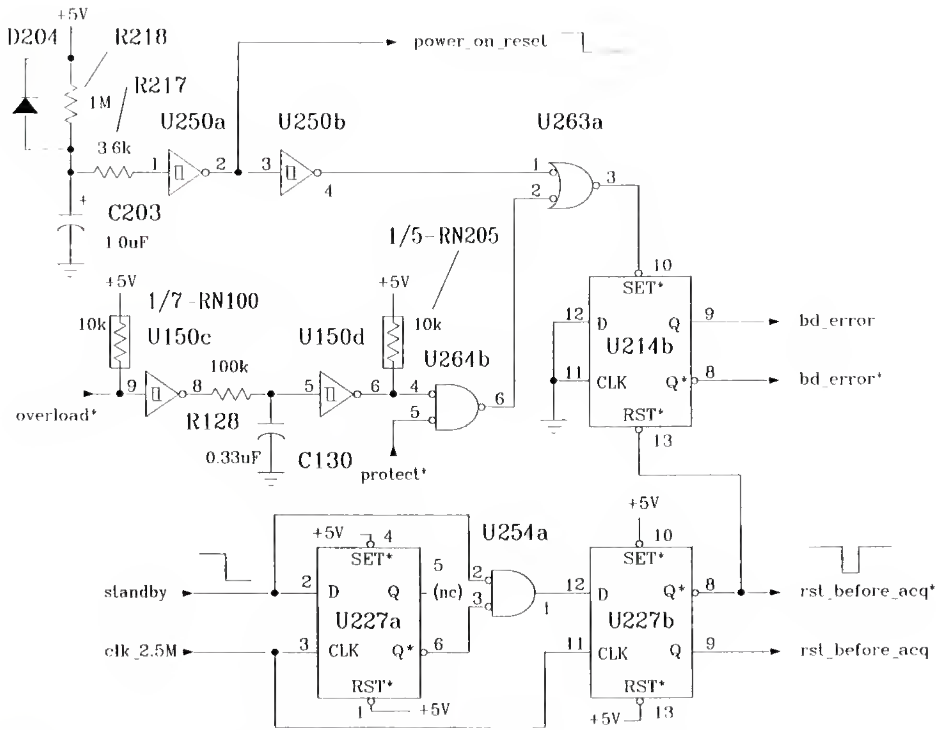
Schematic 4.17 The input signal isolation relay control circuit.



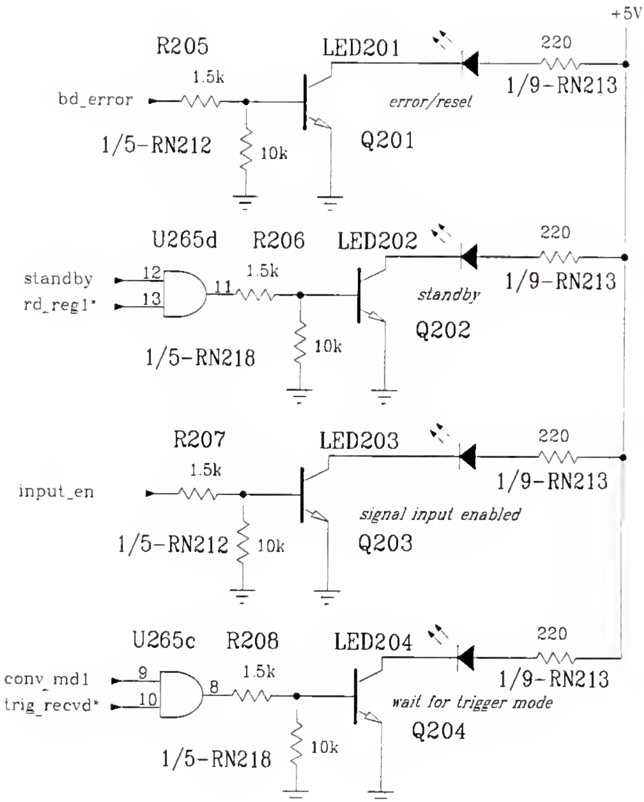
Schematic 4.18 The logic circuitry and drivers associated with the signal routing relays.



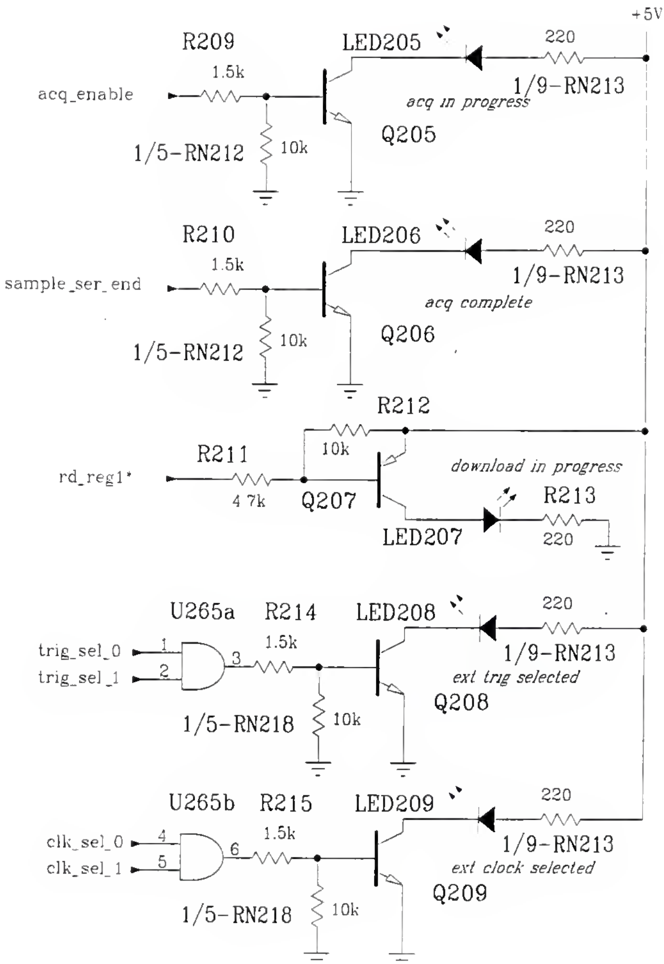
Schematic 4.19 The gain decoder and drivers associated the gain control relays.



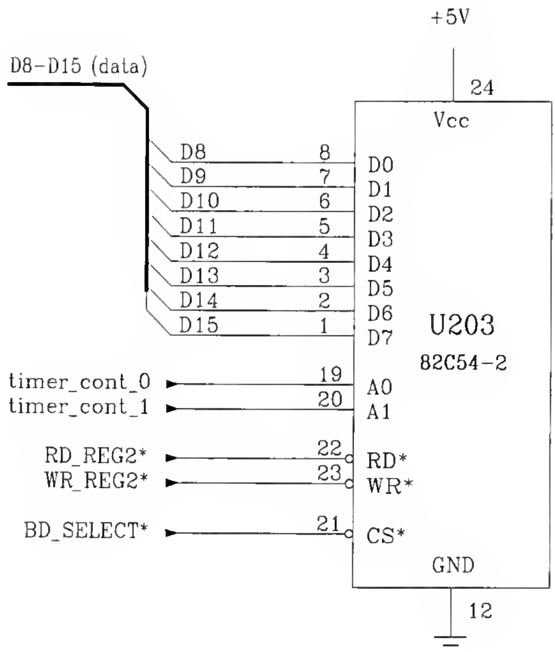
Schematic 4.20 The board-error status register and the reset-pulse generator.



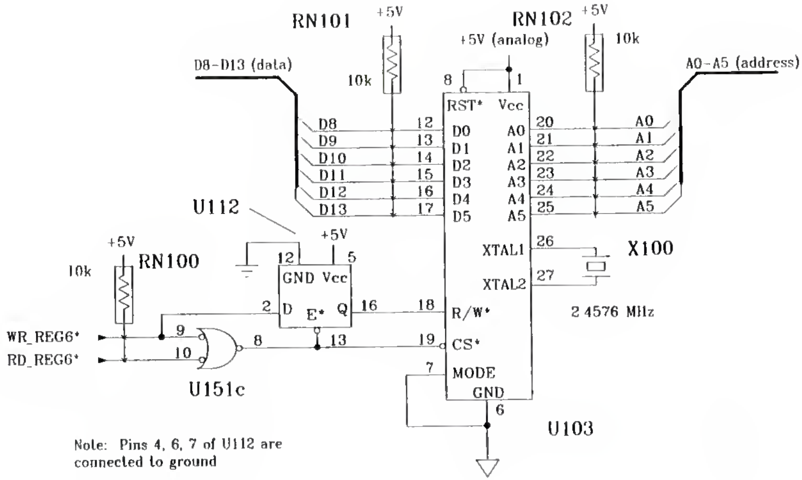
Schematic 4.21a   A/D board front panel LED drivers.



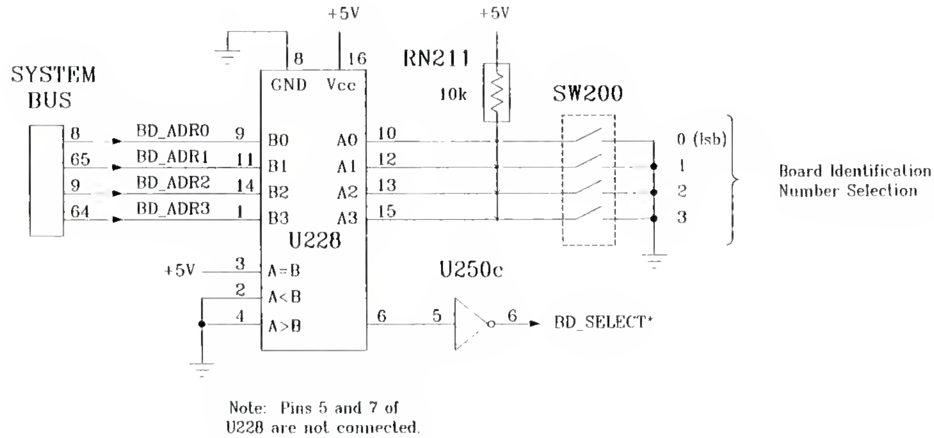
Schematic 4.21b A/D board front panel LED drivers.



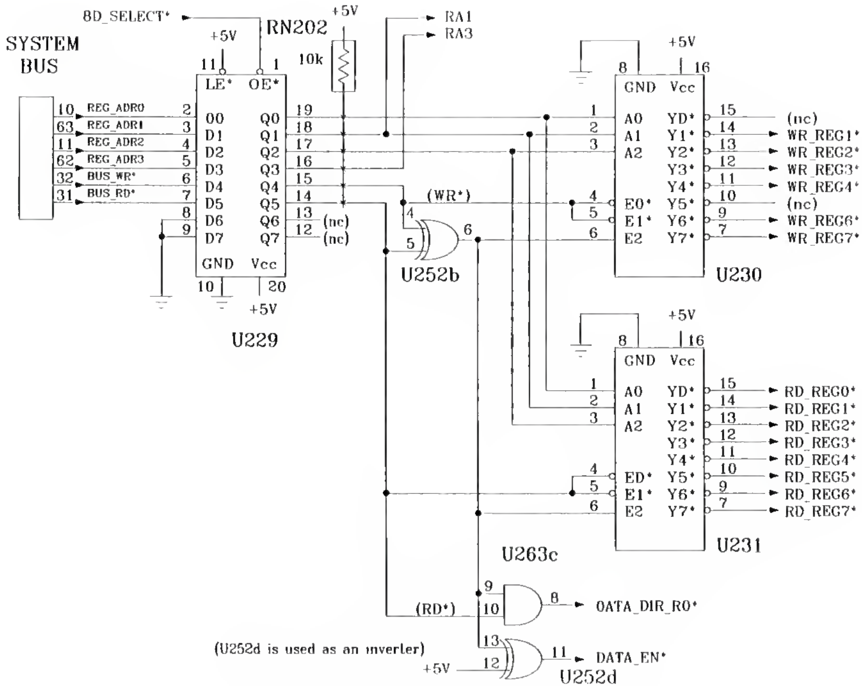
Schematic 4.22 The digital interface for the 82C54-2 triple binary counter.



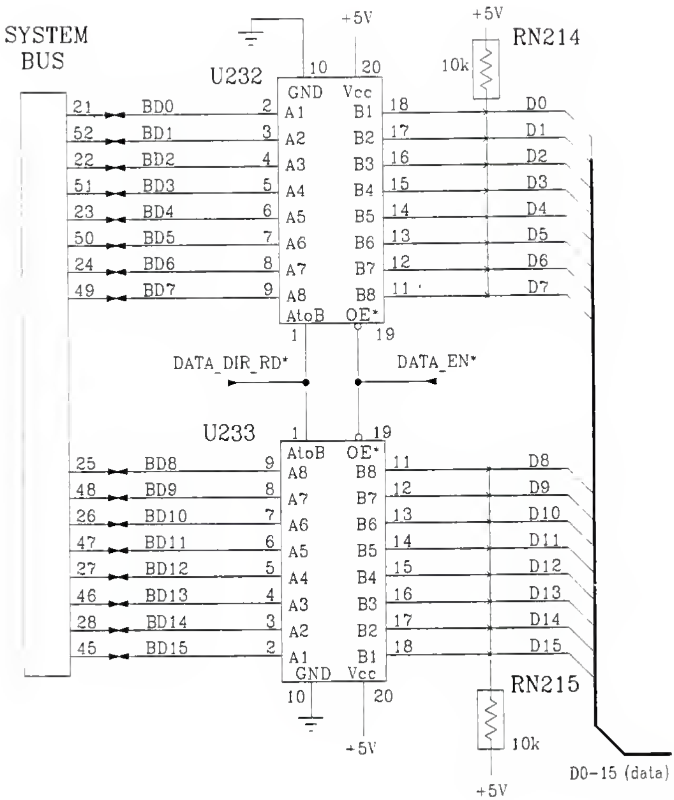
Schematic 4.23 The digital interface for the on-board programmable filter.



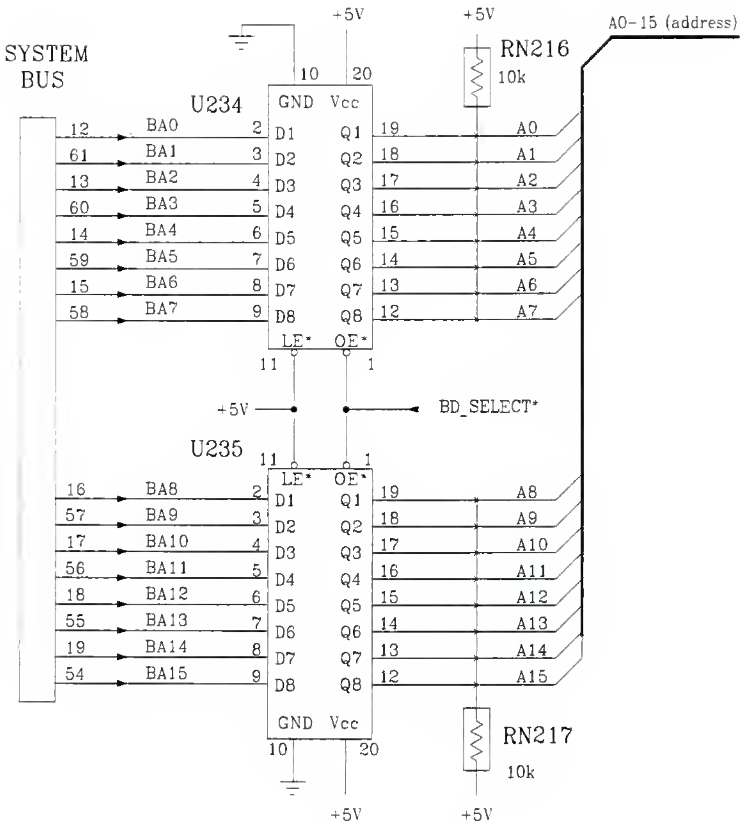
Schematic 4.24 The board address decoder and board address selection circuit.



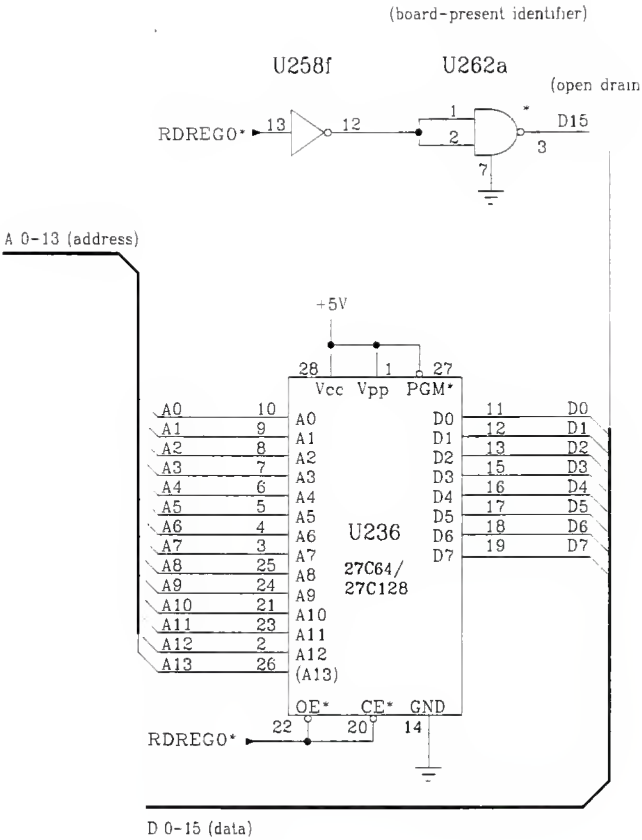
Schematic 4.25 The read/write control logic for the A/D board.



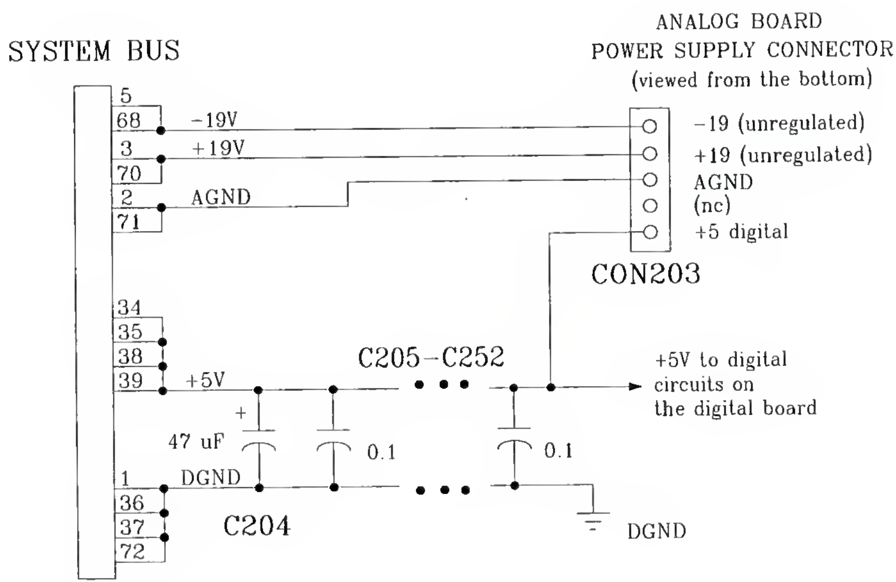
Schematic 4.26 The data buffers between the system data bus and the A/D board data bus.



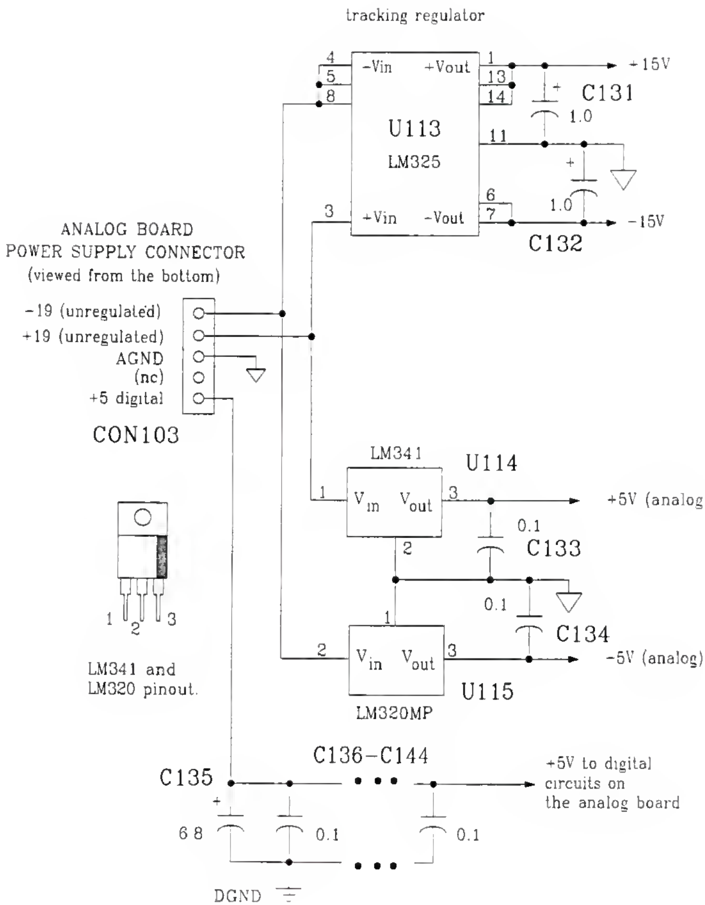
Schematic 4.27 The address buffers between the system data bus and the A/D board 16-bit memory address bus.



Schematic 4.28 The A/D board EPROM circuit.



Schematic 4.29 The bus power connection for the A/D board.



Schematic 4.30 Power supply regulation for the analog signal board.

A/D BOARD PARTS LIST

ID #	part #	description
(note:		all capacitors are 35V, 20% unless otherwise noted)
C101-C104		0.1 uF monolithic capacitor
C105		2000 pF, 50V ceramic (matched with C111)
C106		1000 pF, 50V ceramic (matched with C112)
C107		0.1 uF monolithic capacitor
C108,C109		1.0 uF tantalum
C110		0.1 uF monolithic capacitor
C111		2000 pF, 50V ceramic (matched with C105)
C112		1000 pF, 50V ceramic (matched with C106)
C113,C114		0.1 uF monolithic capacitor
C115		0.01 uF ceramic, 50V
C116,C117		0.1 uF monolithic capacitor
C118		100 pF ceramic, 50V
C119,C120		0.1 uF monolithic capacitor
C121		6.8 uF tantalum
C122		0.1 uF monolithic capacitor
C123		6.8 uF tantalum
C124		0.1 uF monolithic capacitor
C125		6.8 uF tantalum
C126		0.1 uF monolithic capacitor
C127		6.8 uF tantalum
C128		0.33 uF ceramic
C129,C130		0.1 uF monolithic capacitor
C131,C132		1.0 uF tantalum
C133,C134		0.1 uF monolithic
C135		6.8 uF tantalum
C136-C144		0.1 uF monolithic
C201		20 pF ceramic, 50V
C202		(unused)
C203		1.0 uF tantalum
C204		47 uF electrolytic
C205-C252		0.1 uF monolithic capacitor

A/D Board Parts List (continued)

CON101, CON102	right angle flat cable connector (26-pin)
CON103	5-pin Waldom modular connector
CON201,CON202	right angle flat cable connector (26-pin)
CON203	5-pin Waldom modular connector
D101,D102	1N5227 3.6V, 1/2 watt zener diode
D103-D114	1N4148 diode
D200-D203	1N4148 diode
J101	isolated right-angle female BNC connector
J101,J102	chassis mount female BNC connector
JMP101-JMP103	5-station jumper header, 0.1" centers
LED201-LED209	25 mA LED (red)
Q200-Q206	2N2222 general purpose npn
Q207	2N3906 general purpose pnp
Q208	2N2222 general purpose npn

(note: all resistors 1/4 watt, 5% unless otherwise noted)

R101,R102	10 kohm variable resistor
R103	3.65 kohm, 1%
R104	2.0 kohm, 15T linear-taper potentiometer
R105	10 kohm
R106	20 kohm
R107	10 kohm
R108	20 kohm
R109	20 kohm (matched with R118)
R110	12 kohm (matched with R117)
R111	7.5 kohm
R112	24 ohm
R113-R116	5.6 kohm
R117	12 kohm (matched with R110)
R118	20 kohm (matched with R109)

A/D Board Parts List (continued)

R119	7.5 kohm	
R120	10 kohm	linear-taper potentiometer
R121	1.0 kohm	
R122	100 kohm	
R123	3.32 kohm, 1%	
R124,R125	100 ohm, 15T	linear-taper potentiometer
R126,R127	510 kohm	
R128	100 kohm	
R200	2.2 kohm	
R201	100 ohm	
R202	10 kohm	
R203	100 ohm	
R204	10 kohm	
R205-R210	1.5 kohm	
R211	4.7 kohm	
R212	10 kohm	
R213	220 ohm	
R214,R215	1.5 kohm	
R216	2.2 kohm	
R217	3.6 kohm	
R218	1 Mohm	
REL100-REL109	SPST Hg wetted relay, normally open (Clare MSS41A05)	
RN100	9-element, 10 kohm	SIP resistor network
RN101,RN102	7-element, 10 kohm	SIP resistor network
RN200-RN204	9-element, 10 kohm	SIP resistor network
RN205	5-element, 10 kohm	SIP resistor network
RN206-RN209	9-element, 10 kohm	SIP resistor network
RN210	7-element, 10 kohm	SIP resistor network
RN211,RN212	5-element, 10 kohm	SIP resistor network
RN213	9-element, 220 ohm	SIP resistor network
RN214-RN217	9-element, 10 kohm	SIP resistor network
RN218	5-element, 10 kohm	SIP resistor network
SW200	4-STATION DIP SWITCH	

A/D Board Parts List (continued)

U100	AD624C	instrumentation amplifier with pin-programmable gain (Analog Devices)
U101	LM319	dual comparator
U102	AD712	dual op-amp (Analog Devices)
U103	CS7008	switched-capacitor filter + two aux. op-amps (Crystal)
U104	LM311N	comparator
U105	AD346J	sample-and-hold amplifier (Analog Devices)
U106	AD578KN	12-bit analog-to-digital converter (Analog Devices)
U107	74HC221	dual monostable multivibrator
U108, U109	74HC03	quad open-drain 2-input NAND
U110	74HC238	1-of-8 decoder
U111	74HC03	quad open-drain 2-input NAND
U112	74HC75	bistable transparent latch
U113	LM325	±15V tracking regulator
U114	LM320	+5 V regulator, 0.5A, TO-221 style
U115	LM341	-5 V regulator, 0.5A, TO-221 style
U116	741	general purpose operational amplifier
U150	74HC14	hex Schmitt-trigger inverter
U151	74HC08	quad 2-input AND
U152	74HC02	quad 2-input NOR

(note: U116-U149 are unused part numbers)

U200	74LS04	hex inverter for oscillator
U201	74HC4040	clock divider
U202	74HC153	dual 4-to-1 MUX
U203	82C54-2	triple 16-bit counter
U204	74HC153	dual 4-to-1 MUX
U205	74HC74	dual D Flip-flop, rising edge trig.
U206		(unused)
U207	74HC74	dual D Flip-flop, rising edge trig.
U208, U209	74HC574	octal clocked latch
U210, U211	74HC245	octal data transceiver
U212, U213	CD4520	dual 4-bit binary counter
U214	74HC74	dual D Flip-flop, rising edge trig.

A/D Board Parts List (continued)

U215,		
U216,		
U217,		
U218	74HC573	octal transparent latch
U219,		
U220	74HC574	octal clocked latch
U221,		
U222,		
U223,		
U224	43256	32Kx8 SRAM, 150 ns version (CMOS)
U225	82C55A	parallel peripheral interface
U226	74HC139	dual 1-of-4 decoder
U227	74HC74	dual D Flip-flop, rising edge trig.
U228	74HC85	4-bit magnitude comparator
U229	74HC573	octal transparent latch
U230,		
U231	74HC138	1-of-8 decoder
U232,		
U233	74HC245	octal data transceiver
U234,		
U235	74HC573	octal transparent latch
U236	27C64	8Kx8 EPROM (150 ns version)
U250	74HC14	hex Schmitt-trigger inverter
U251	74HC08	quad 2-input AND
U252	74HC86	quad 2-input exclusive-OR
U253	74HC00	quad 2-input NAND
U254	74HC02	quad 2-input NOR
U257	74HC11	triple 3-input AND
U258	74HC04	hex inverter
U262	74HC03	quad 2-input open-drain NAND
U263	74HC08	quad 2-input AND
U264	74HC32	quad 2-input OR
U265	74HC08	quad 2-input AND

(note: U206, U237-U249, U255, U256, U259-U261 are unused part numbers)

X100	2.4576 MHz crystal (HC-18)
X200	10 MHz crystal (HC-18)

Interfigure signals for A/D board schematics

Signal name	Description	Schematic from	to
A 0-15	16-bit memory address present on A/D board	4.27	4.12 4.22 4.28
ACQ_ENABLE	active when all conditions for data acquisition are satisfied e.g. desired triggers have been received, acquisition series has not been completed	4.8	4.21b
A/D_CONV*	pulse when conversion is to occur; conversion occurs at the end of the pulse (rising edge). The width of this pulse is determined by counter #2 of the 82C54-2 (U203).	4.8	4.4 4.7 4.11
A/D_DAT 0-11	12-bit 2's compliment value from A/D converter	4.4	4.10
A/D_EOC	Active when analog-to-digital converter has completed making a conversion	4.4	4.9 4.10 4.15
BA 0-15	16-bit system bus memory address, TTL-level	BUS	4.27
BD 0-15	16-bit system bus data, TTL-level	BUS	4.26
BD_ADR 0-3	4-bit, TTL-level, specifies which board is being addressed	BUS	4.24
BD_ERROR	an inverted version of BD_ERROR*	4.20	4.4 4.17 4.21a
BD_ERROR*	active for one of two reasons: 1) the +5V digital power has been recently applied; 2) signal overload has been sensed. This signal is reset by the transition from "standby" mode to a "non-standby" mode e.g. "conv_now."	4.20	4.8 4.15
BD_SELECT*	active when the board address on the bus matches the value set by the board address DIP-switch	4.24	4.15 4.23 4.25 4.27
BUS_CLK	TTL-level clock available on the bus	BUS	4.5
BUS_RD*	bus read strobe (active low), TTL-level	BUS	4.25
BUS_WR*	bus write strobe (active low), TTL-level	BUS	4.25
CLK_2.5M	A 2.5 MHz clock signal (1/4 the crystal frequency): 400 ns period	4.5	4.9 4.20

Interfigure signals for A/D board schematics (cont)

Signal name	Description	Schematic from	to
CLK_5.0M	A 5.0 MHz clock signal (1/2 the crystal frequency): 200 ns period	4.5	4.8
CLK_SEL_0	the least-significant-bit of clock select	4.15	4.5 4.21b
CLK_SEL_1	the most-significant-bit of clock select	4.15	4.5 4.21b
CONV_MD_0	the least-significant-bit of conversion mode	4.15	4.16
CONV_MD_1	the most-significant-bit of conversion mode	4.15	4.7 4.16 4.21a
CONV_NOW*	"convert now" mode when active	4.16	4.7 4.8
CONV_ON_TRIG*	"convert on trigger" mode when active	4.16	4.8
CONV_PRE_TRIG*	"convert with pre-trigger sample retention" mode when active	4.16	4.8
D 0-15	16-bit data present on A/D board	4.26	4.10 4.12 4.15 4.22 4.23 4.28
DATA_BUF_EN*	active when data buffer may be enabled; this is when RD is not equal to WR and the board is selected.	4.25	4.26
DATA_DIR_RD*	sets the direction of the bus data buffer, determined by the bus RD* signal	4.25	4.26
EN_A/D_BUF*	enables latched A/D converter data to be read by on-board memory (during conversion sequence) or the system bus (read REG 7)	4.9	4.10
EN_MEM_BUS*	enables the on-board memory to data bus transceiver	4.9	4.10
FLTR_OUT*	disengages filter from analog signal path when active	4.15	4.18
GAIN_0	the least-significant-bit of gain selection	4.15	4.19
GAIN_1	gain selection	4.15	4.19
GAIN_2	the most-significant-bit of gain selection	4.15	4.19

Interfigure signals for A/D board schematics (cont)

Signal name	Description	Schematic from	to
IA_OUT	single-ended analog signal from the instrumentation amplifier	4.1	4.2
INPUT_EN	active when the signal input relays are engaged	4.17	4.21a
MA 0-15	16-bit on-board sample memory address	4.11 4.12	4.13 4.14
MD 0-15	16-bit on-board sample memory data	4.10	4.13 4.14
MEM_CYCLE*	active when the sample memory has recycled while waiting for the trigger	4.11	4.15
MEM_EN	enables on-board memory during a memory read or memory write operation	4.9	4.13 4.14
OVERLOAD*	pulled low when an overload is sensed by the I.A. overload sense circuit	4.1	4.20
POST_FILTER	the analog signal following on-board filtering	4.2	4.3
POWER_ON_RESET	a short pulse active at power-up	4.20	4.15
PROTECT*	active enables overload protection	4.15	4.20
RA 1, 2	buffered versions of the bus register address lines, used for selecting registers on the control/status register	4.25	4.15
RD_REG0*	active during a read from register 0, the configuration EPROM and the board-present identification bit	4.25	4.28
RD_REG1*	active during a read from register 1, the on-board memory	4.25	4.9 4.13 4.14 4.21a 4.21b
RD_REG2*	active during a read from register 2, the interval-timer/counter chip	4.25	4.23
RD_REG3*	active during a read from register 3, status register (con/stat configure register)	4.25	4.15
RD_REG4*	active during a read from register 4, control register #1 (control #2)	4.25	4.15
RD_REG5*	active during a read from register 5, the trigger address register	4.25	4.12

Interfigure signals for A/D board schematics (cont)

Signal name	Description	Schematic from	to
RD_REG6*	active during a read from register 6, the on-board filter	4.25	4.22
RD_REG7*	active during a read from register 7, the latched data from the A/O converter	4.25	4.9
REG_ADR 0-3	4-bit, TTL-level, specifies which register is being addressed	BUS	4.25
RST_BEFORE_ACQ	a 400 nsec pulse that occurs prior to an acquisition sequence	4.20	4.11
RST_BEFORE_ACQ*	an inverted version of RST_BEFORE_ACQ	4.20	4.6 4.11
SAMP_SER_END	an inverted version of SAMP_SER_END*	4.7	4.21b
SAMP_SER_END*	active when the desired number of post-trigger samples has been acquired	4.7	4.8 4.15
SIG_RT_1*	"1st" output of signal route decoder	4.16	4.17
SIG_RT_3*	"3rd" output of signal route decoder	4.16	4.18
SIG_SEL_0	the least-significant-bit of signal selection	4.15	4.16
SIG_SEL_1	the most-significant-bit of signal selection	4.15	4.16
STANDBY	active when operating mode is set to "standby"	4.16	4.9 (2) 4.11 4.20 4.21a
STANDBY*	an inverted version of STANDBY	4.16	4.9 (2) 4.12
TIME_FOR_SAMPLE	active at the end of the sample-period counting series indicating anther sample should be taken	4.5	4.8
TIMER_CONT_0	the least-significant-bit of the counter address	4.15	4.23
TIMER_CONT_1	the most-significant-bit of the counter address	4.15	4.23
TO_SHA	analog signal selected for SHA input	4.3	4.4
TRIG_BUS	TTL-level signal which serves as system trigger	BUS	4.6
TRIG_RECVD	an inverted version of TRIG_RECVD*	4.6	4.7
TRIG_RECVD*	active when the selected trigger is received; is reset by RST_BEFORE_ACQ*	4.6	4.8 4.15 4.21

Interfigure signals for A/D board schematics (cont)

Signal name	Description	Schematic from	to
TRIG_RISE*	trigger circuit is sensitive to rising edge when active	4.15	4.6
TRIG_SEL_0	the least-significant-bit of trigger selection	4.15	4.6 4.21b
TRIG_SEL_1	the most-significant-bit of trigger selection	4.15	4.6 4.21b
TRIG_SIG	makes the appropriate transition as the signal amplitude traverses the trigger level	4.3	4.6
WR_MEM*	pulse when data is to be latched into on-board memory; the data may originate from the A/D converter or the system bus	4.9	4.13 4.14
WR_REG1*	register 1, the on-board sample memory	4.25	4.9 4.10
WR_REG2*	active during a write to register 2, the interval-timer/counter chip	4.25	4.23
WR_REG3*	active during a write to register 3, status register (con/stat configure register)	4.25	4.15
WR_REG4*	active during a write to register 4, control register #1 (control register #2)	4.25	4.15
WR_REG6*	active during a write to register 6, the on-board filter	4.25	4.22
WR_REG7*	active during a write to register 7, used to initiate a conversion by the A/D converter	4.25	4.8

CHAPTER FIVE

SUGGESTED SYSTEM CONTROLLER ALGORITHMS

5.1 Introduction

The purpose of this chapter is to describe the software for the system controller and the generic I/O board. Since the scope of this thesis does not include a detailed account of the system front-end control algorithms, only a proposal will be made concerning how these might be organized. The feasibility of the algorithms listed in this chapter is unknown -- they are provided as a starting place for those who develop the actual routines.

As mentioned in Chapters 2 and 3, the board selected for the system controller was the Motorola 68HC11EVB: an evaluation board for the M68HC11 microcontroller. This board is ideally suited for the task of system controller since it is equipped with: an RS-232 communication port, on-board ROM and RAM, and accessible address and data bus. The algorithms described in this chapter enable the 68HC11EVB to receive commands from the host via the RS-232 link, decipher the commands, dispatch these commands to the appropriate board, and, at the completion of these steps, send a return message to the host computer.

An excellent source of information for algorithm implementation on the 68HC11EVB is the M68HC11EVB Evaluation Board User's Manual. When purchased, the 68HC11EVB was supplied with the BUFFALO ("Bit User's Friendly Aid to Logical Operation") monitor program. This program contains many useful routines that might be useful for the system controller algorithms described later in this chapter. Chapter 3 of the 'EVB manual contains information about these routines, and the 'EVB manual's Appendix B has the complete source code for the BUFFALO monitor program. Review of this material is highly recommended before undertaking the composition of the system controller algorithms.

A good programming reference for the M68HC11 is M68HC11 HCMOS Single-Chip Microcontroller Programmer's Reference Manual. This book provides information concerning the M68HC11 instruction set, as well as information about its addressing modes.

The material from this chapter is presented in the following manner. First, the duties of the system controller are fully described. This includes each of the routines that make up the system controller software. Second, the software specifications for an I/O board are presented. This includes suggestions for the arrangement of the program stored on the board's EPROM.

5.2 Algorithm Format for the System Controller

As far as the host computer is concerned, the data acquisition system behaves much like any peripheral device connected to an RS-232 port e.g. a plotter or printer, where the communication sequence between the host computer and the peripheral is much like the following:

1) the host computer sends a command to the peripheral in the form of ASCII characters;

2) once the termination character is received, the peripheral deciphers the command and takes the appropriate action (execution of the appropriate routine);

3) the peripheral returns an acknowledgement of the receipt of the command. If the command sent by the host requested the peripheral to return information, this acknowledgement would contain the information requested.

Since the system controller is responsible for all communication with the host computer, it is responsible for all of the duties of the peripheral listed above. This is made possible if the system controller is a finite-state machine, where some of its duties include:

1) receive an ASCII command string from the host (via the RS-232 communication link);

2) compute a check sum for the received command and take appropriate actions if an error exists;

3) delegate command to the appropriate command module;

4) return a message to the host;

5) copy a board's command set from the board's EPROM into system controller memory.

The system controller algorithms are all implemented on a 68HC11EVB. The memory allocation of this 8-bit microcomputer board is listed in Table 5.2.1.

Table 5.2.1 Memory allocation for the 68HC11EVB (following modifications listed in Appendix E).

Description	Address
Internal RAM (MCU reserved)	\$0000 - \$00FF
Not used	\$0100 - \$0FFF
(reserved)	\$1000 - \$17FF
Not used	\$1800 - \$3FFF
Flip-Flop decode	\$4000 - \$5FFF
Optional RAM (8K)	\$6000 - \$7FFF
Not used	\$8000 - \$97FF
Terminal ACIA	\$9800 - \$9FFF
Bus driver ports	\$A000 - \$AFFF
Not used	\$B000 - \$B5FF
EEPROM	\$B600 - \$B7FF
Not used	\$B800 - \$B7FF
RAM (8K)	\$C000 - \$DFFF
System controller EPROM (8K)	\$E000 - \$FFFF

As shown in Table 5.2.1, 8K of memory space is available for the system controller routines, and a total of 16K of RAM is available for other needs.

An important aspect of the system controller is the use of the system controller's memory. Fig. 5.2.1 illustrates one possible allocation of the system controller memory. Functions which pertain to the system controller are stored in the ROM (EPROM). These routines include host-communication, the command dispatcher, memory management, system bus control, and system controller command implementation. The system controller RAM is where all board routines are stored, as well as the location where communication queues are maintained.

Fig. 5.2.2 shows the arrangement of the system controller routines, and the remainder of this section discusses the initialization of the system, the command-handling routines, and other assorted routines that are performed by the system controller.

System controller reset/initialization

When power is freshly applied to the 68HC11EVB (or when the "reset" push button is pressed), the system is reset and the routine pointed to by the power-up vector ($FFFE-$FFFF) is executed. The system controller ignores all commands until it receives a command instructing the system to initialize (this is the "si" command listed in Appendix B).

The system-initialize command causes the system to prepare itself for a data acquisition session. This consists of the following steps being taken:

1) Initialize the memory control parameters;

2) Configure the bus driver control ports as specified in section 3.4 (initialization of the bus drivers);

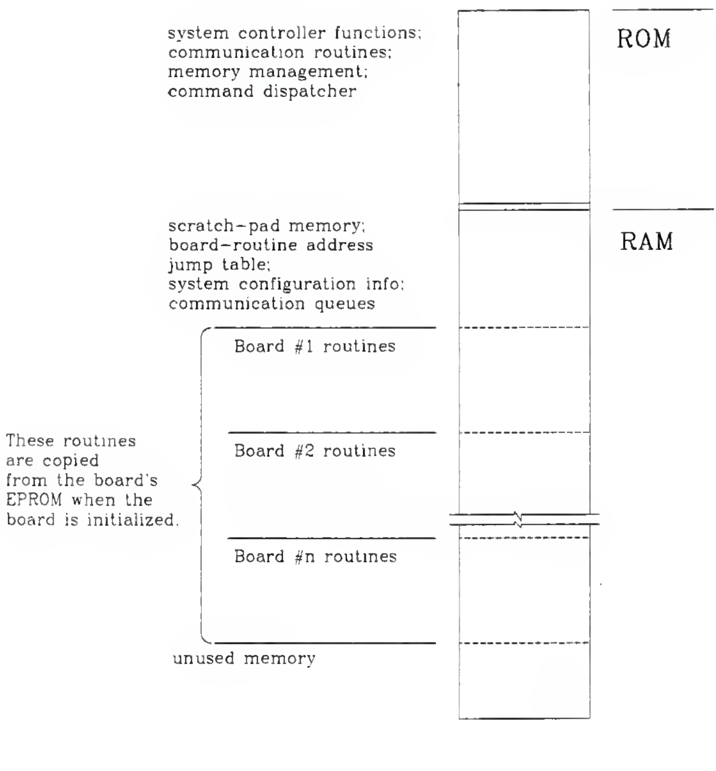


Figure 5.2.1 The allocation of system controller ROM and RAM.

3) Return a "system initialized" message to the host.

At the completion of this initialization routine, the system controller is ready to receive and execute commands.

The message receiver

Characters are received from the host by way of the RS-232 communication link joining the host computer with the DAS. When a character is received from the host, an interrupt is generated, thus causing the sequence of operations shown in Fig. 5.2.3(a) to be executed. The received character is retrieved from the RS-232 "received character" register and is stored in the "received-message queue". The received character is compared with the message termination character (;) and the "message end" status flag is set accordingly. The routine then returns from the interrupt.

The command dispatcher

As shown in Fig. 5.2.3(b), the system controller waits for a command to be sent, and the receipt of a complete command is indicated when the "message end" status flag goes active. The system controller then calculates a check sum value for the received command and compares it with the check sum received as part of the message. If the check sums do not compare favorably, the

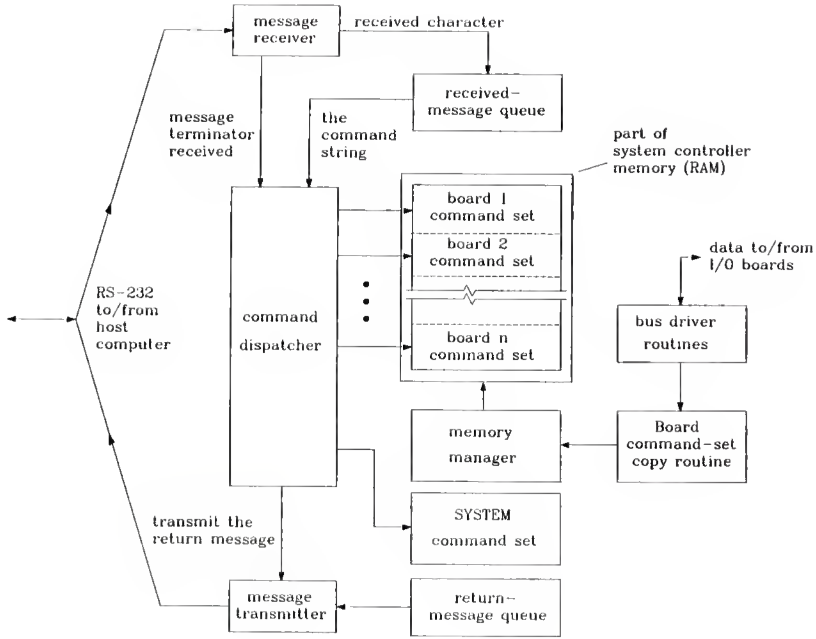
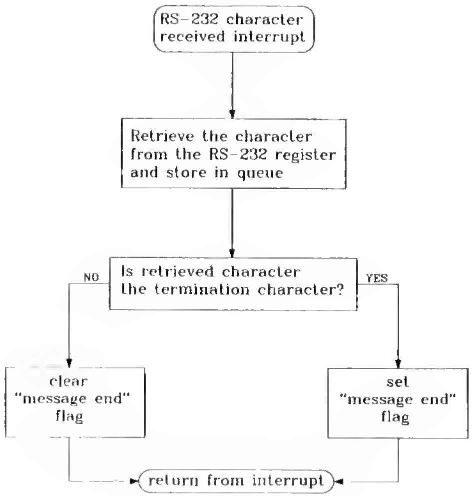


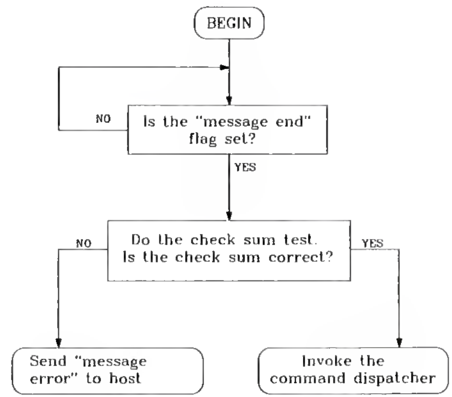
Figure 5.2.2 The software modules which comprise the system controller.

system controller returns a "transmission error" message to the host, otherwise, the command is deciphered by the command dispatcher.

The operation of the command dispatcher is simply a matter of executing the routine which corresponds with the command received from the host. As specified in Appendix B, the command's identity is determined by the first two characters, and if necessary, the board address is the number following these two letters. As shown in Fig. 5.2.4, the routine corresponding to the command is found by first searching the list of system controller commands, and, if necessary, searching the command list of the specified board. Once the command is found, its associated routine is executed as a subroutine by the command dispatcher. At the completion of the routine, control is returned to the command dispatcher. The command dispatcher then signals the "message transmitter" to transmit the return message to the host computer.



(a)



(b)

Figure 5.2.3 Operations performed by the system controller: (a) the interrupt routine which handles a character when it is received, and (b) waiting for a character to be sent from the host.

Return message transmission

During the execution of a command implementation routine, messages to be sent to the host computer are stored in the "return message queue". When signalled by the command dispatcher, the message transmitter sends the message in the return message queue to the host. As with messages sent to the DAS, each message sent to the host is terminated by a semi-colon (;). After the return message has been sent, the system controller waits for another command from the host. Note: Contents of the return-message queue preserved until a routine stores a fresh return message -- this facilitates the capability of a message resend.

Utility routines and subroutines

Numerous routines are also made available as a part of the system controller. These routines include:

Bus driver routines. The bus driver routines facilitate bus-related I/O with the boards. These routines manage data to/from I/O-board registers, and all memory-, register-, and board-related addressing.

Board initialization. Before an I/O board can be used during a data acquisition session, it must be initialized. The following steps are performed

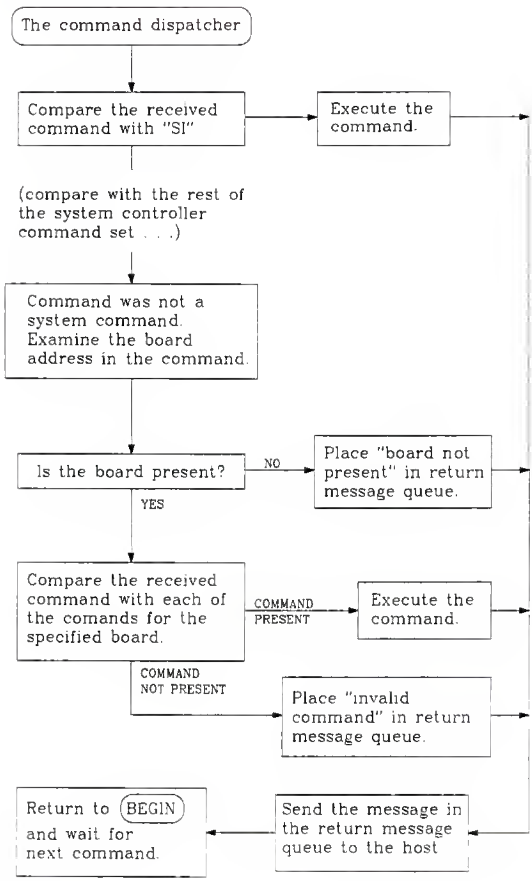


Figure 5.2.4 The sequence of operations performed by the command dispatcher.

during board initialization:

1) Verify the board is present [A board is present if D_{15} of register 0 is LOW].

2) Read the first two bytes of the board's EPROM and determine the memory requirements for this board's routines [EPROM data is read from the LSB of register 0].

3) If sufficient system controller memory is available, the system controller sets up a place in its RAM and invokes the "board command-set copy routine" i.e. the board's EPROM is copied into system controller RAM.

4) Upon completion of the copy routine, the board's own initialization routine is executed (this was one of the routines that was just read from the EPROM).

5) Report the successful board initialization to the host.

Memory management. The purpose of the memory management routine is to supervise the allocation of the system controller's RAM. The memory manager is responsible for positioning an I/O-board's routine set when it is copied from the board's EPROM. In addition, the memory manager must maintain a board-routine jump-address table.

Other subroutines. Other subroutines are included within the system controller that may be used by the I/O-board command implementation routines. These include BCD-to-binary conversion, binary-to-BCD conversion, and other routines which may be useful to many boards.

5.3 I/O-Board Command Implementation

The purpose of this section is to present a set of guidelines for I/O-board command implementation. These guidelines include rules for software and organization on the I/O-boards' EPROMs, and some assembly language techniques that might be useful when composing the I/O-board programs.

One way to minimize the complexity of the system controller software is to require that all I/O-board command-implementation routines to be organized in the same manner. The following rules are suggested for each of the I/O -board routines.

RULE 1: All coding in the board's EPROM start at address 0000, and the first two bytes of this code indicate the length of the code stored in the EPROM.

A method is needed to inform the system controller the length of the code for a particular board. This permits the routine which copies the code to determine two things: 1) if there is sufficient room for the code in the system controller memory, and 2) when to stop copying the code during the copy routine.

RULE 2: All code written for the I/O-board must be relocatable.

Rule 2 is necessary since the contents of the board's EPROM are copied into system controller RAM at a position that is unknown at the time of code compilation.

RULE 3: The command look-up table for each board must begin in a standard position (for example, beginning at the first byte following the "length of code" information mentioned in RULE 1).

This rule simplifies memory management and command dispatch routines.

RULE 4: All buffers, variables, and constants for a particular board must reside within the EPROM-code address bounds.

Any memory that a board needs for variables, scratch-pad memory, or other applications must be addressed within that board's code-space. This rule reduces the complexity of the memory manager that would otherwise be responsible for allocating, protecting, and deallocating the memory between the boards. Although this rule is not particularly memory efficient, it does reduce the complexity of the memory management routine.

RULE 5: Routines for an I/O board may NOT alter memory outside of its code-space.

This is a natural extension of rule 6: limiting the addressable space for each board to its own code-space reduces the complexity of the memory manager.

RULE 6: A command for a board may not have the same two-letter identification as any of the system controller commands. However, boards may share two-letter command identifiers.

As described in section 5.2, the command dispatcher assumes that the received command is a system controller command, and only after an unfavorable survey of the system controller commands does the dispatcher determine the address of the board to which the command is intended.

The I/O-board EPROM memory map

Another way that system controller software complexity can be reduced is to require the placement of software on the I/O-board's EPROM to be stored in a standardized manner. One possible way is shown in Fig. 5.3.1. The following material in this section describe the I/O-board command organization shown in Fig. 5.3.1.

End-address identifier. The purpose of the end-address identifier permits the system controller to know the space required to load the I/O-board's code, and so the system controller knows the last address it needs to copy during the EPROM-copy routine. The method of identifying the

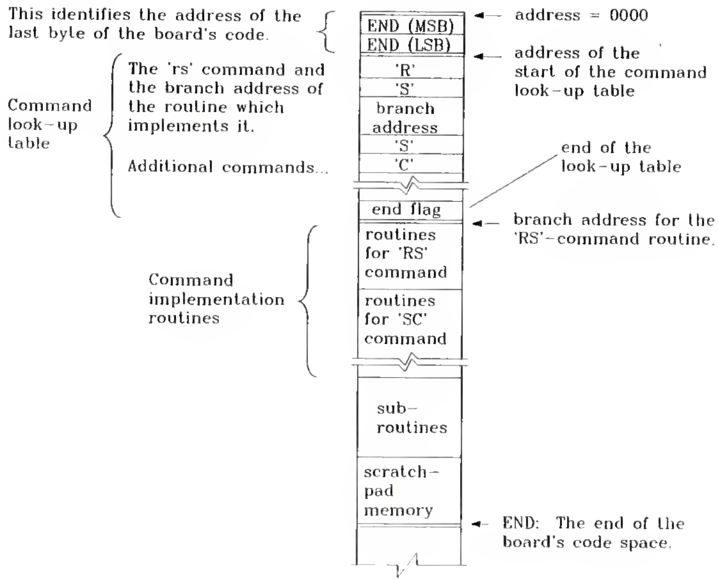


Figure 5.3.1 The code space for an I/O-board, as it is arranged on the board's EPROM.

last address could be done in many ways. One way is to use an FDB compiler instruction which places the 16-bit "end" address within the assembly code at compile time. This assembly code might appear as follows:

```
            ORG     $0000           *  All code is stored
                                    *  in the EPROM
                                    *  beginning at address
                                    *  0000.

            FDB   #END_BYTE         *  The 16-bit address
                                    *  of the last byte
                                    *  of the assembly
                                    *  code is placed in
                                    *  the first two bytes
                                    *  at compile time.

                  :         :
                  :         :

END_BYTE    EQU       *             * The last byte
                                    * of code.
```

For the case shown above, the system controller reads the 16-bit number at EPROM address 0, where this 16-bit number is the address of the last byte of code.

Command look-up table. The second section of the I/O-board code space is the command look-up table. This table is accessed when a command for the board is received. The system controller's command dispatcher compares the two ascii characters in the look-up table with the command received. If a match is made, the routine pointed to by the "branch address" is executed as a subroutine. The end of the command look-up table is marked by a special "end flag" value to alert the command dispatcher that the command look-up table has ended. An example of the assembly code for the command look-up table is shown below.

```
        FCC   'rs'          *  "Retrieve sample"
                            *  command identifier.
        FDB   #RET_SAMP     *  The branch address
                            *  of "retrieve
                            *  samples" routine
        FCC   'sc'          *  The next command
                            *  and its
        FDB   #ST_CONV      *  branch address.

         :       :
         :       :

        FDB   #END_TABLE    *  The end-of-table
                            *  identifier.
```

<u>Command implementation routines</u>. The next section of the I/O-board's code space is the routines which implement the commands. Each of these routines are called as subroutines by the command dispatcher, where the branch address was given in the command look-up table. An example of the assembly code for the command implementation routines is shown below.

```
RET_SAMP   EQU   *          *  The branch address
                            *  of the RET_SAMP
                            *  routine is defined.

           :     :          *  The code for the
           :     :          *  routine.
           RTS              *  Return from the
                            *  subroutine.
ST_CONV    EQU   *          *  The branch address
                            *  of the next routine.
           :     :
           :     :
```

Subroutines. Routines that are shared by the command implementation routines are included in the section called "subroutines". The routines included in this section are useful for this particular I/O board, but are not included in the routines resident in the system controller subroutines.

Scratch-pad memory. This section of the I/O-board's code space is the region reserved for storing information pertaining to the board, or used for any other general purpose variable storage. The size of this memory space may be large or small, depending upon the needs of the I/O board.

CHAPTER SIX

SUMMARY

A design has been presented for a host-independent data acquisition system. This acquisition system has the advantage of being usable by virtually any computer that has an RS-232 port. The motivation for the system's development was to reduce the cumbersome interface problems frequently encountered when moving an acquisition system from one host computer to another.

This thesis has described a prototype of a system that is modular in the sense that it is composed of a system controller and up to sixteen removable I/O boards, all of which are interconnected by a system bus. The system controller receives mnemonic commands from the host computer by way of an RS-232 communication link, whereupon the command is deciphered and the appropriate actions are taken. An important goal during the design of this system was to minimize complexity of the host computer's system-controlling software, thus reducing the development time when the system is used with a new host.

In addition, the design of an analog-to-digital conversion board for use with the system was presented. Among this board's numerous features is its 150 kHz sampling rate and its 64K sample on-board memory.

A good deal of work remains for those who compose and implement the controlling algorithms for the system. These algorithms will determine the ease with which the system can be used. The DAS design presented in this thesis places as few restrictions as possible on those who develop the controlling algorithms, and those who design and construct additional I/O boards which are compatible with the system.

REFERENCES

1. M68HC11EVB Evaluation Board User's Manual. Phoenix, AZ: Motorla, Inc., 1986.

2. Crystal 1988 Data Book. Austin, TX: Crystal Semiconductor, p. 9-10.

3. Intel Microsystem Components Handbook, Volume II (1986). Santa Clara, CA: Intel Corporation, p. 6-335.

4. Crystal 1988 Data Book. Austin, TX: Crystal Semiconductor, p.9-3/9-21.

5. Data Conversion Products Databook (1988). Norwood, MS: Analog Devices, p. 6-5.

6. Linear Products Databook (1988). Norwood, MS: Analog Devices, p. 4-56.

7. Data Conversion Products Databook (1988). Norwood, MS: Analog Devices, p. 3-84.

APPENDIX A

USING THE DAS WITH SYSTEM CONTROLLER SIMULATOR

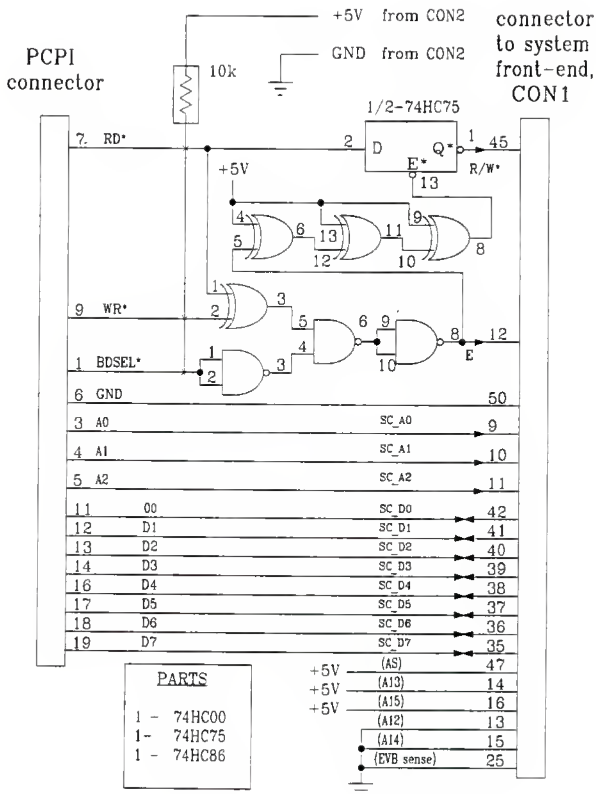
A.1. Introduction

The purpose of this appendix is to provide instructions to the user of the Data Acquisition System (DAS). This includes setting up the system and using it with the PC-interface. In addition, this Appendix includes instructions for making adjustments on the A/D board.

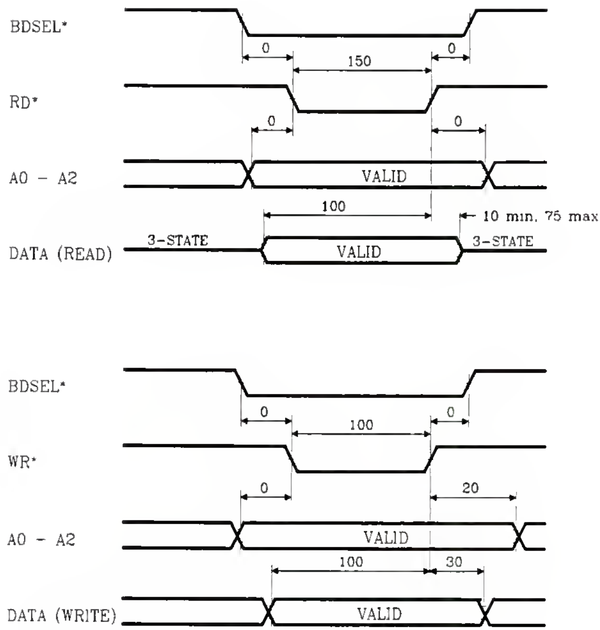
A.2. Simulating the System Controller

Since the system controller (68HC11EVB) was not implemented with this system, some means of testing the system without the controller was necessary. Therefore, a PC equipped with an 8-bit parallel interface (PCPI) was used to simulate the controller board [1]. In addition to the data bits, this interface had the appropriate number of control lines for the bus driver circuit. The PC also had a C-language source code compiler thus enabling system test routines to be implemented.

In order to use the PCPI with the system front-end, a small interface circuit was needed. This circuit is shown in Schematic A.1. The operation of this circuit is straight forward: a small circuit simulates the E and R/W* signals from the 68HC11EVB based on the three data control lines from the PCPI (BD_SEL*, RD*, and WR*). Additionally, address lines A0 - A2 provide bus driver port selection. The timing diagram for the PCPI-to-bus driver interface is shown in Fig. A.2.1.



Schematic A.1 The schematic for the PCPI-to-DAS interface circuit.



Notes: 1. These drawings are NOT TO SCALE.
2. All times are in nanoseconds, and should be reguarded as minimums.

Figure A.2.1 The necessary timing when the bus driver circuit is used with the PCPI and the interface circuit shown in Schematic A.1.

A.3. Using the System with the PC Controller

Using the system is simply a matter of following these instructions:

1. <u>Setting up the system</u>

A. Verify all power to the system is OFF. If the PC power is on, execute the PI_DAS program and disable the interface.

2. Carefully install the desired boards into the card sockets on the system front-end board.

3. Connect the 26-conductor ribbon cable from the PCPI board (in the PC) to the system controller interface board.

4. Connect the power supply to the system with banana connectors on the side of the system base-board. All power should be energized simultaneously. The Hewlett-Packard 6236B triple output power supply in one possible power supply candidate. Although this supply is only capable of 7V (for the +8V source), it will work with this system.

> **IMPORTANT:** Before installing or removing a board from the system, make sure the system power is OFF!

5. Execute the PI_DAS.EXE program on the PC.

6. The system must now be initialized. Type "si" for system initialization. This routine will enable the PCPI board interface, prompt the user for the board address of the A/D board, and initialize the

board. The address of the A/D board is set with the 4-station DIP switch, SW200 (see Fig. D.2.2, Appendix D).

The system is now ready to be used.

2. <u>Using the system test routines</u>

The commands recommended for the system (listed in Appendix B) are implemented in the test program, PI_DAS.EXE. The commands available from the test program are listed in Tables A.1 through A.5. Please refer to Appendix B for descriptions of these routines.

Table A.1 PCPI/DAS: Routines exclusive to the DAS/PCPI test routine.

Command	Description
;	Comment. Type the comment immediately following the ';'.
help	Presents a help screen for the user of the PCPI_DAS.
gv	Get value produced by A/D conversion -- select source, initiate conversion, then display the value.
l	Toggle session logger "on" or "off."
mt	Memory test for on-board memory.
sb	Set board address to which all board related commands are routed. This command is automatically executed during system initialization.
quit	Terminate session and close the session-log file.
step	Toggle the step-by-step execution of bus read/write operations.
trace	Toggle the screen dump of bus read/write operations.

Table A.2 PCPI/DAS commands: System controller.

Command	Description
si	SYSTEM INITIALIZE
bi	BOARD INITIALIZE
ct	CONFIGURE SYSTEM TRIGGER
cc	CONFIGURE SYSTEM CLOCK
bw <reg>	WRITE TO BOARD REGISTER
br <reg> <value>	READ FROM BOARD REGISTER
dr	DISPLAY BUS DRIVER PORT VALUES

Table A.3 PCPI/DAS commands: A/D Board commands for conversion control configuration.

Command	Description
ts	TRIGGER SELECT
cs	CLOCK SELECT
fs	FULL SCALE SIGNAL RANGE
fi	FILTER CONTROL
fc	FILTER CONFIGURE
sr	SET SAMPLE RATE
cl	CALIBRATE

Table A.4 PCPI/DAS commands: A/D Board commands for conversion control.

Command	Description
bc	BEGIN CONVERSION
sc	STOP CONVERSION

Table A.5 PCPI/DAS commands: A/D board status query and data retrieval.

Command	Description
gs	GET STATUS
rs	RETRIEVE SAMPLES

Table A.6 PCPI/DAS commands: auxiliary commands for test program.

Command	Description
d	Disable PCPI interface.
e	Enable PCPI interface.
i	Toggle both initialization lines -MRESET and -BDINIT either "on" or "off."
ia	Activate -MRESET and -BDINIT.
ib	Activate -BDINIT.
id	Deactivate -MRESET and -BDINIT.
im	Activate -MRESET.
p	Plot contents of data buffer on graphics display.
pp	Plot contents of data buffer on pen plotter.
r <reg>	Read the contents of register <reg>, where <reg> = 0, ... , 7.
w <reg> <value>	Write the value <value> to register <reg>, where <reg> = 0, ... , 7. and <value> can be given in decimal (default), binary, octal, or hexadecimal. Example: A decimal 23 can be entered as Decimal: 23 or 23d Binary : 10111b Octal : 27o Hex : 17h

Each of these commands simulate the commands described in Appendix B. The only difference is that commands implemented by the test program prompt the user for all necessary parameters (the command structure given in Appendix B require the parameters to be part of a command string).

Many commands from the original PCPI control program are also available with the PCPI_DAS control program. These commands are listed in Table A.6.

3. <u>A description of a data acquisition session</u>

The following is a key-by-key description of a typical session with the DAS in conjunction with the PCPI and Zenith PC.

1. Turn the power to the computer ON, and turn the power ON to the Selanar graphics terminal.
2. From the C:\E747\DDN directory, execute the pi_das program.
3. Once the pi_das program prompts for a command, turn the power ON to the DAS prototype circuit.

   Note: To receive assistance with commands available with the prototype test circuit, type help at the command prompt. This will display a list of the commands with a brief description of each.

4. Issue the "initialize system" command, si, and answer the prompt concerning the board address based on the board-presence survey at the beginning of the initialize system routine.

   Note: If power to the prototype circuit needs to be disconnected following the previous step, the "disable interface" , d, command should be issued.

The remaining steps in this description are for data acquisition with the A/D board: 1) setting up the acquisition parameters, 2) initiating the acquisition sequence, and 3) retrieving and displaying the acquired data.

5. Setting up the A/D board acquisition parameters:

   Selecting the clock: type cs and select the appropriate clock source. In this simple example, select the internal clock followed by an acquisition rate of 10000 Hz. The frequency set on the A/D board is then displayed on the PC monitor. (Note: the sample rate may also be set by the sr command.)

   Selecting the trigger: type ts and select the appropriate trigger.

   Selecting the full-scale range of the input amplifier: type the "full-scale" command, fs, followed by the desired gain. For this example, select gain = 1.

6. Initiating the acquisition sequence.

Connect the signal source to the signal-input BNC-connector on the A/D board front-panel. The signal source for this example might be a periodic function with a fundamental frequency an order of magnitude less than the sampling frequency e.g. a 4 V_{p-p} 100 Hz sine wave.

Type the "begin conversion" command, **bc**, and answer the prompts. For this example, use the Immediate conversion mode, which means conversion begins as soon as all prompts in this command are answered. Answer the number of samples with 200, and use signal input protection.

The status of the board may be obtained in two ways: 1) observing the LEDs on the A/D board front panel, or 2) executing the "get status", gs, command.

7. Retrieving the acquired data.

As soon as the A/D board has completed the data acquisition sequence, the data is ready to be transferred from the A/D board memory to the host computer. This is accomplished in the following manner:

(1) return the A/D board to the "standby" condition by issuing the "stop conversion", sc, command.

(2) retrieve the samples by issuing the "retrieve samples", rs, command. This will tell how many samples were acquired and prompt for the number of samples to be retrieved . . . answer with 200, and begin with the first sample. Once the samples have been retrieved, save the data to a disk-file.

(3) display the acquired data on the Selanar graphics terminal. Issue the "plot", p, command, and plot all 200 points. The data should then appear on the graphics terminal.

This example sequence is complete. Type quit to exit the program. Once the PC returns to the DOS prompt, turn the power to the prototype circuit OFF.

A.4. Calibrating the A/D board

The purpose of this section is to delineate the procedure for calibrating the A/D board. The specific adjustments that are made available to the user are:

Instrumentation Amplifier:

input offset null for I.A.
output offset null for I.A.
x10 gain adjustment

Analog-to-Digital Converter:

bipolar offset
gain adjust

The specific instructions for each of these adjustments is detailed in the following sections. For each adjustment, full power should have been applied to the system for one half hour, while the system is at the same temperature of the environment which the measurements will be made. Please refer to Fig. A.4.1 for location of test points, adjustments, etc. when using the following instructions.

<u>Adjustments pertaining to the Instrumentation Amplifier</u>

Input offset null:

1. At the host computer, set the board's mode to STANDBY by issuing the stop conversion (sc) command.

2. Connect one lead of a DVM to TP100 and the other lead to TP101. Adjust the DVM to the lowest DCV scale available (at least 2 mV full scale).

3. Referring to Fig. A.4.1, place a jumper on JMP101 and JMP103.

4. Referring to Fig. A.4.1, adjust R101 until the magnitude of the voltage measured by the DVM is minimized.

5. Remove the jumpers and place them back on JMP 104 and JMP105. Remove the DVM connectors.

Output offset null:

1. At the host computer, set the board's mode to STANDBY by issuing the stop conversion (sc) command.

2. Connect one lead of a DVM to TP100 (next to the I.A.) and the other lead to TP101. Adjust the DVM to the lowest DCV scale available (at least 2 mV full scale).

3. Referring to Fig. A.4.1, place a jumper on JMP101 and JMP102.

4. Referring to Fig. A.4.1, adjust R102 until the magnitude of the voltage measured by the DVM is minimized.

5. Remove the jumpers and place them back on JMP 104 and JMP105. Remove the DVM connectors.

x10 gain adjustment:

1. From the host computer, issue the calibrate (cl) command.

2. Obtain a steady voltage source with an output between 250 mV and 500 mV; measure its output voltage; connect it to the A/D board signal input.

3. Connect a DVM to TP100 and TP101 and adjust the scale of the DVM to permit measurement of 2 vdc to 10 vdc.

5. From the host, execute the "calibrate" routine, select the "x10 gain adjust routine".

6. While monitoring the DVM, adjust R104 until the meter reads exactly ten times the value of the signal source.

7. Remove the DVM and cancel the calibration routine (press the <enter> key).

Adjustments pertaining to A/D converter [2]:

1. From the host, execute the calibrate (cl) routine.

2. Obtain a stable voltage source that can be adjusted to 4.9988 volts and 4.9963 volts. Connect this source to the signal input for the A/D board.

3. Connect a 5-1/2 digit DVM to TP102 and TP100.

NOTE: The "positive" lead of the DVM must be connected to TP102. Set the DVM's scale to measure approximately 5 volts.

4. From the host, execute the "A/D converter adjustment" routine.

5. While, watching the DVM, adjust the input signal until the voltage is exactly -4.9988 volts (if positive voltage is being measured, reverse the

polarity of the voltage source: restart this adjustment routine from step 1).

6. While watching the host computer's monitor, adjust R124 until the follow transition occurs:

10000000 00000000 ----> 10000000 00010000.

7. From the host computer, turn the calibrate routine off by responding "0" to the number of conversions prompt.

8. Reverse the polarity of the voltage source.

9. While, watching the DVM, adjust the input signal until the voltage is exactly +4.9963 volts (if negative voltage is being measured, reverse the polarity of the voltage source: restart this adjustment routine from step 7).

10. While watching the host computer's monitor, adjust R125 until the follow transition occurs:

01111111 11100000 ----> 01111111 11110000.

11. From the host computer, cease the calibrate routine.

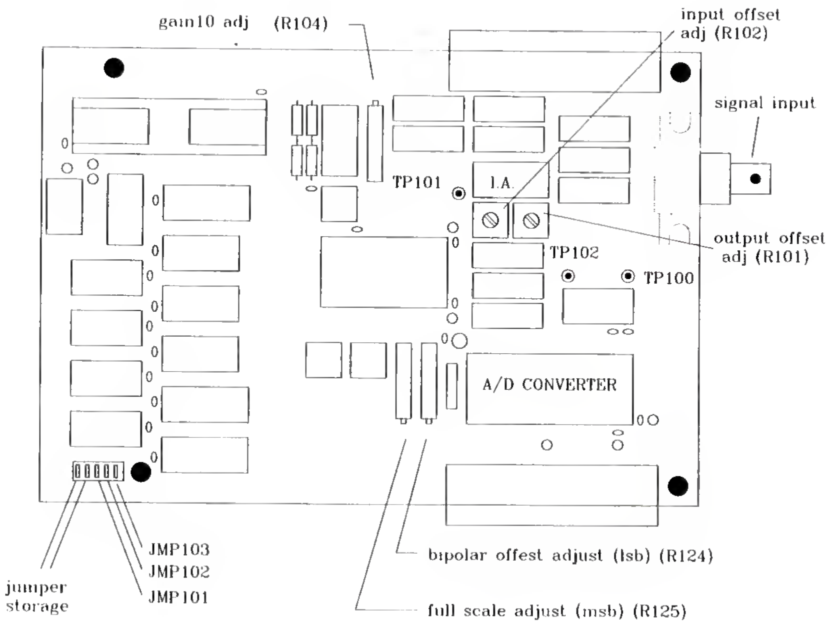


Figure A.4.1 View of the A/D board for location of calibration components.

APPENDIX A REFERENCES

1. S. Dyer, "The PCPI - A Personal Computer Parallel Interface," Dept. of Electrical and Computer Engineering, Kansas State University, (unpublished) 1989.

2. Analog Devices, Data Conversion Products Handbook 1988, p. 3-84.

APPENDIX B

THE PROPOSED COMMAND SET FOR THE DAS

The purpose of this appendix is to present the proposed command set for the final version of the DAS. Whether these commands are used or not, the material in this appendix lists some of the commands that might be useful.

The material in this appendix is presented in the following manner. Following a summary of the commands, an explanation for the command-format is presented. The remainder of the appendix is a detailed list of the commands.

Table B.1 presents a summary of the proposed commands.

Table B.1 Data acquisition system command summary.

System Controller Commands
general purpose commands
NULL COMMAND
SYSTEM INITIALIZE
RESEND LAST MESSAGE
BOARD INITIALIZE
CONFIGURE/ACTIVATE SYSTEM TRIGGER
CONFIGURE SYSTEM CLOCK
BOARD REMOVE
board testing commands
BOARD REGISTER WRITE
BOARD REGISTER READ
A/D Board Commands
conversion configuration/system calibration
TRIGGER SELECT
CLOCK SELECT
FULL SCALE SIGNAL RANGE
FILTER CONTROL
FILTER CONFIGURE
SET SAMPLE RATE
conversion control
BEGIN CONVERSION
STOP CONVERSION
board status query/data retrieval
GET STATUS
SEND SAMPLES
GET DATA HEADER

Command Format

Each of the commands are composed of 7-bit ASCII characters in the following format:

command (board number) [opt1, ..., opt n] terminator

where

command	is composed of exactly two alphabetic characters (A-Z);
(board number)	is the decimal characters for the board to whom the command is directed . . . this is omitted when the command is directed to the system controller;
[options]	are command-related parameters (the use of which is functionally dependant) . . . all parameters are separated by commas (,);
terminator	A semi-colon (;) terminates all commands (both to and from the system).

All spaces (ASCII 32 decimal) within commands are ignored.

Each of the commands listed in Table B.1 are described throughout the remainder of this appendix in the format of the following command description.

An example command description . . .

BI (n);	<===	instruction format (as sent by user character-by-character)
Command:	BOARD INITIALIZE	<== command name
Purpose:	This instructs the system controller to load the routines from the board's ROM into system controller RAM.	
Parameters:	n is the number identifying the board.	
Return:	ACK; normal return	<=== (character-by-character response from DAS)
	BNP; error, board not present;	
	IM; error, insufficient system controller memory to load the board's command set.	

The proposed command set

NOTE: In the event of an error, the system may return the following code as a response to any command:

NACK;	error in transmission, possible extraneous characters were received prior to the receipt of the (;)

System controller functions

;

Command:	NULL COMMAND
Purpose:	This command has no purpose other than to flush the system's message que e.g. to reset the received-message buffer in the system controller. This command is useful if a command has been sent but the system fails to respond. If the system is failing to respond due to a communication error that occurred during the transmission of the failed command, the null command should forces the system to make some form of return to the host, and consequentially clear the message que (thus making the system ready for a new command).
Return:	ACK;

SI;

Command:	SYSTEM INITIALIZE
Purpose:	This instructs the system controller to initialize the bus and format the controller's RAM.
Return:	ACK;

RM;

Command:	RESEND LAST MESSAGE
Purpose:	This instructs the system controller to resend the most previous message sent to the host computer.
Return:	[The most previous message];

BI **bd_num;**

Command:	BOARD INITIALIZE
Purpose:	This instructs the system controller to load the routines from the board's ROM to system controller RAM.
Parameters:	**bd_num** is the number identifying the board.
Return:	ACK; if board initialization went without flaw,
	BNP; error: board not present,
	IM; error: insufficient system controller memory to load the board's command set.

CT control;

Command:	CONFIGURE/ACTIVATE SYSTEM TRIGGER
Purpose:	This instruction configures the bus trigger (from the system controller). Additionally, this routine permits set-up of the external system trigger.
Parameter:	control = D for trigger deactivation; SR pre-set for RISING-edge trigger activation; SF pre-set for FALLING-edge trigger activation; A for activation of system controller trigger activation; CE connect to EXTERNAL trigger source; CI connect to INTERNAL trigger source.
Return:	ACK; normal return.

CC source;

Command:	CONFIGURE SYSTEM CLOCK
Purpose:	This instruction selects between the internal 3.0 MHz clock and the external clock connection on the front panel of the controller.
Parameters:	source = I for internal, E for external.
Return:	ACK;

BW bd_num, register, value;

Command:	WRITE TO A SPECIFIC REGISTER ON A BOARD
Purpose:	This command enables a register on a board to be written to directly. This command is especially useful during board testing. This function does not verify the board is present before writing.
Parameters:	**bd_num** is the board's number, 0-15, **register** is the register to which the value is to be written, 0-15, **value** is the number to be written to the specified register, 0-65535 (16-bit range).
Return:	ACK;

BR bd_num, register;

Command:	READ THE CONTENTS OF A SPECIFIC REGISTER ON A BOARD
Purpose:	This command enables a register on a board to be read from directly. This command is especially useful during board testing. This function does not verify a board is present before reading.
Parameters:	bd_num is the board's number, 0-15, register is the register from which the value is to be read, 0-15.
Return:	The number read from the register (in decimal ASCII format).

A/D board commands

The following commands are proposed for use with the A/D board.

TS bd_num, source, edge;

Command:	TRIGGER SELECT
Purpose:	Select the appropriate trigger and assign the appropriate signal edge: rising or falling. The sources include the board's front panel trigger, the bus trigger, or the signal (in the same manner as an oscilloscope). The edge may be rising or falling.
Parameters:	bd_num is the board's number, 0-15, source = B for Bus, P for panel, S for signal, SB for signal with simultaneous bus trigger activation, edge= R for rising edge, F for falling.
Return:	ACK; normal return, BNP; error: board not present.

CS bd_num, source;

Command:	CLOCK SELECT
Purpose:	Select the appropriate clock for conversion. Selections for source include the bus clock , and the board's front panel clock input. The clock signal employed by the board is slowed down by a counter, where the counter value is set by the count parameter.
Parameters:	bd_num is the board's number, 0-15, source = B for Bus, P for Panel.
Return:	ACK; normal return, BNP; error: board not present, PE; error: parameter error.

FS bd_num, gain value;

Command: FULL SCALE SIGNAL RANGE

Purpose: Set the voltage gain of the board's front-end amplifier, depending upon the instrumentation amplifier selected.

Parameters: bd_num is the board's number, 0-15, gain value may be 5V, 500MV, 50MV, 25MV, 10MV.

Return: ACK; normal return,

BNP; error: board not present,
PE; error: parameter error.

FI bd_num, control;

Command: FILTER CONTROL

Purpose: This instruction inserts or removes the on-board anti-aliasing filter.

Parameters: bd_num is the board's number, 0-15, control = I for "filter in", O for "filter out".

Return: ACK; normal return,

BNP; error: board not present,
PE; error: parameter error.

FC bd_num, base_addr, n, byte_1, byte_2, ... , byte_n;

Command: FILTER CONFIGURE

Purpose: Set up the on-board anti-aliasing filter with the given coefficients. The order of the bytes is not known at this time. (Note: if non-consecutive filter registers are to be written to by this configuration instruction, one may use adjacent commas with non character between, i.e.

. . . byte_1,byte_2,,byte_4,byte_5 . . .

the third register was skipped.)

The configuration data for the filter is contained in byte_1 through byte_n. These configuration data must be in the following format:

MSB LSB

1	D5	D4	D3	D2	D1	D0

where D5 through D0 compose the six-bit configuration data. The most significant bit must be set (this most significant bit is ignored).

Parameters: bd_num is the board's number, 0-15,
base_addr is the filter register address in which byte_1 is stored,
n is the number of bytes in the configuration,
byte_i are the n-bytes sent to the filter.

Return:

ACK; normal return,

BNP; error: board not present,
PE; error: parameter error.

SR bd_num, period;

Command:	SET SAMPLE RATE
Purpose:	Set the sample clock register to provide the specified sampling rate. The period of the sampling rate must be an integer.
Parameters:	**bd_num** is the board's number, 0-15, **period** is the sampling frequency period in milliseconds.
Return:	ACK; normal return,
	BNP; error: board not present,
	PE; error: parameter error.

BC bd_num, cont, #_pre_trigger_samples, #_post_trigger_samples;

Command:	BEGIN CONVERSION
Purpose:	This instruction initiates conversion. The control parameter selects between "immediate conversion" or "wait for trigger". In addition, this instruction sets the pre- and post- trigger configuration. The board will determine whether or not the configuration is legal i.e. whether or not the total number of samples is less than or equal than board memory.
Parameters:	bd_num is the board's number, 0-15, cont is I for immediate, W for wait for trigger. #_pre_trigger_samples is the number of pre-trigger samples, #_post_trigger_samples is the number of post-trigger samples requested, (note: the sum of post- and pre- trigger samples must not exceed the number of samples on the board. Behavior of the board when given erroneous parameters is unknown.)
Return:	ACK; normal return, BNP; error: board not present, PE; error: parameter error.

SC bd_num;

Command:	STOP CONVERSION
Purpose:	This instruction stops conversion immediately. Samples taken prior to receipt of this instruction are retained on board.
Parameters:	bd_num is the board's number, 0-15.
Return:	ACK; normal return, BNP; error: board not present.

GS bd_num;

Command: GET STATUS

Purpose: This instruction retrieves the status of board **bd_num**. Status is returned in the format specified by that board. A description of the status message follows this list of functions.

Parameters: **bd_num** is the board's number, 0-15.

Return: The ASCII header as shown on the page entitled "Status Query",

BNP; error: board not present.

RS bd_num, nth_sample, m_samples;

Command: RETRIEVE SAMPLES

Purpose: This instruction requests m data sample data be sent, starting with the nth-sample. In the case of pre-triggering, retrieval of all data requires m = pre + post, where pre and post are the number of samples taken for each case. Also, in the case of pre-triggering, the 1st sample is the first sample taken <u>before</u> receipt of the trigger.

Parameters: **bd_num** is the board's number, 0-15, **nth_sample** is the starting place in the sample series for the retrieval, **m_samples** is the number of samples to be retrieved.

Return: The return sequence begins with

ACK,$samp_1$,$samp_2$, ... ,$samp_n$;

Each of the samples are in decimal format separated with commas (,) and the entire sequence is terminated with a semi-colon (;).

BNP; error: board not present,
PE; error: parameter error.

GH bd_num;

Command: GET DATA HEADER

Purpose: Instructs the board to send the ascii data header.

Parameters: bd_num is the board's number, 0-15.

Return: ACK; normal return,

BNP; error: board not present.

STATUS QUERY

When the GET STATUS command is sent to an A/D board, information for each of the parameters listed below are returned to the host. The information will be the appropriate option listed below.

<u>parameter</u>	<u>options</u>
ACQUISITION MODE:	1. Standby 2. Waiting for trigger 3. Waiting for trigger w/pre-trigger acquisition 4. Acquisition in progress 5. Acquisition complete 6. BOARD ERROR -- INPUT MAY BE OVERLOADED
TRIGGER SOURCE:	1. Signal level = xxxx % full scale 2. Signal level with simultaneous bus trigger activation 2. External (front panel of board) 3. Bus
TRIGGER SELECT:	1. Positive (rising) edge 2. Negative (falling) edge
CLOCK SELECT:	1. Internal 2. External (front panel of board) 3. Bus (front panel of system controller)
SIGNAL INPUT:	1. Isolated--input removed 2. Input selected
SAMPLE RATE:	(the sampling period and frequency of the internal clock)
ON-BOARD FILTER:	1. Filter in circuit 2. Filter removed from circuit
FULL SCALE SIGNAL INPUT RANGE:	1. 5 V 2. 500 mV 3. 50 mV 4. 25 mV 5. 10 mV
PROTECTION:	1. Protection enabled 2. protection disabled

APPENDIX C

DESIGN OF ADDITIONAL BOARDS FOR THE DAS

C.1. Introduction

One of the most important design objectives of this system is that additional boards can be designed, constructed and used with the system with a minimum of difficulty. The purpose of this appendix is to delineate the procedure that should be followed during the design of a board to be used with this system.

As noted in Chapter 2, the generic I/O board can be divided into three parts: the bus interface, the control/status register, and the board specific circuitry. The design of the bus interface and control/status registers used for the A/D board design (presented in Chapter 4) should suffice for most designs.

C.2. The Board Design Procedures

Given that a special purpose board is to be constructed for use with the system, the following procedure should be followed.

1. decide on a list of specifications for the board;
2. compose a list of commands to be used with the board;
3. design the board-specific circuitry;
4. generate high-level algorithms that manage the board-specific circuitry in order to implement the commands;
5. determine the circuitry and logic necessary to control the board specific circuitry, keeping in mind the commands associated with the board;
6. determine the signals required to both monitor and control the board;
7. make register assignments to the bus I/O registers;
8. after making refinements to the high-level command algorithms (taking into account concessions made during the design of the control circuitry) convert them into assembly code for the system controller (in the language appropriate for the system controller microprocessor);
9. "burn" the assembly code into an EPROM;
10. construct the board;
11. test the board with the register read/write operations (from the system command set) to verify operation of algorithms;

12. Install the EPROM and test the board and make corrections to the command algorithms on the EPROM if errors exist.

All boards used with the system must have a bus interface to connect to the system bus. The pin out of this bus is shown in Table C.1. This bus interface must perform several things, including board address comparison, register I/O address decoding, address/data buffering, and provide an EPROM at register 0. The remainder of this appendix presents a recommended circuit to be used as the bus interface for a board.

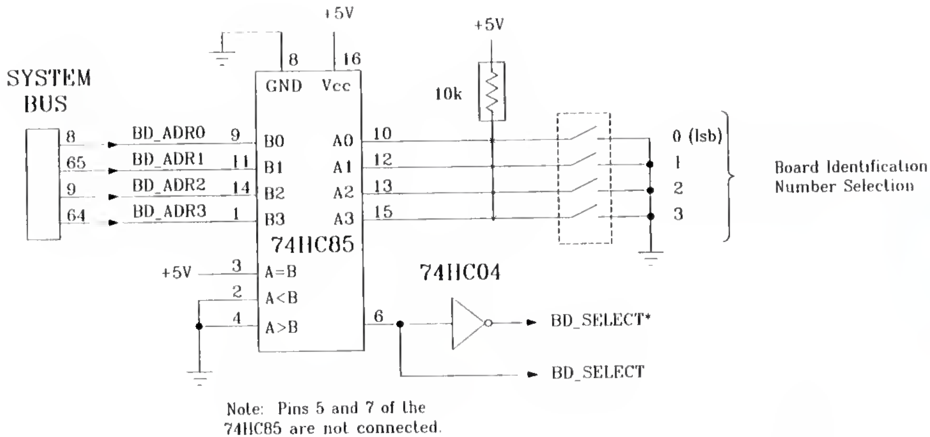
The board's address and address recognition circuit is shown in Schematic C.1. A 4-bit identity comparator in conjunction with a 4-station DIP switch permits the board address on the bus to be decoded. When the board address is recognized, BD_SELECT becomes active.

An 8-register I/O decoding circuit is shown in Schematic C.2. All register address lines are buffered and then provided to a pair of 1-of-8 decoders (one for write and one for read requests). Additional circuitry shown in this circuit is for the data buffer controls.

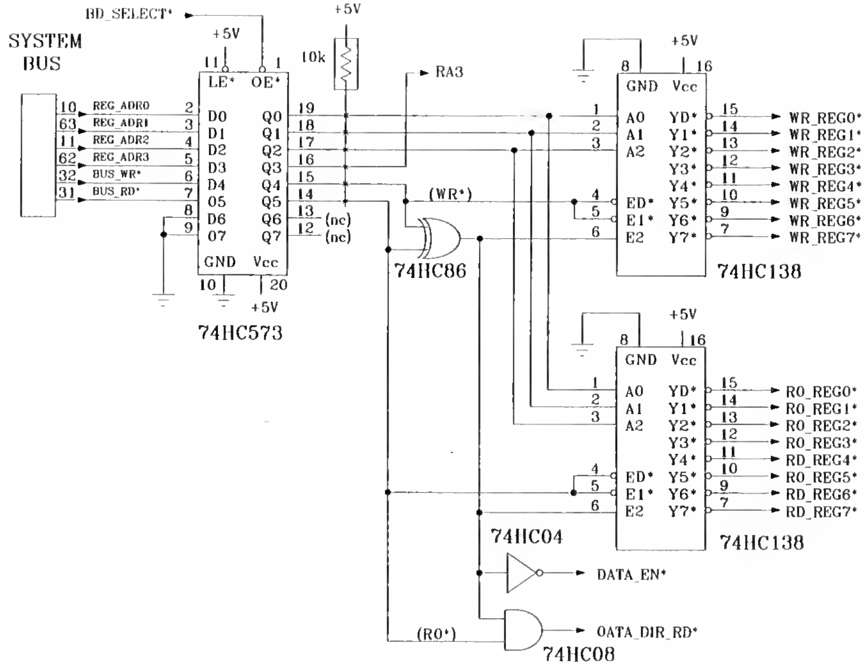
Schematics C.3 and C.4 are the circuits used to buffer the 16-bit bus address lines and the data lines. The board's EPROM is connected as shown in Schematic C.5, and the board's power is retrieved from the bus as shown in Schematic C.6.

Table C.1 The pin out of the system bus, as viewed from the connecter edge.

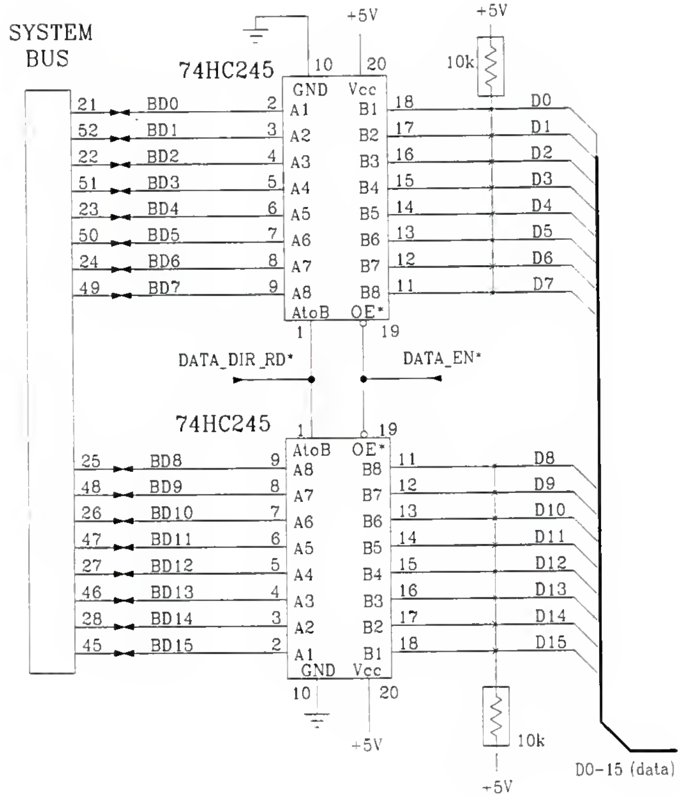
DGND	1	72	DGND
AGND	2	71	AGND
+19V	3	70	+19V
AGND	4	69	AGND
-19V	5	68	-19V
AGND	6	67	AGND
DGND	7	66	DGND
BD_ADR0	8	65	BD_ADR1
BD_ADR2	9	64	BD_ADR3
REG_ADR0	10	63	REG_ADR1
REG_ADR2	11	62	REG_ADR3
BA0	12	61	BA1
BA2	13	60	BA3
BA4	14	59	BA5
BA6	15	58	BA7
BA8	16	57	BA9
BA10	17	56	BA11
BA12	18	55	BA13
BA14	19	54	BA15
DGND	20	53	DGND
BD0	21	52	BD1
BD2	22	51	BD2
BD4	23	50	BD4
BD6	24	49	BD6
BD8	25	48	BD8
BD10	26	47	BD10
BD12	27	46	BD12
BD14	28	45	BD14
DGND	29	44	DGND
BUS_CLK*	30	43	(reserved)
RD*	31	42	(reserved)
WR*	32	41	BUS_TRIG*
DGND	33	40	DGND
+5V (REG)	34	39	+5V (REG)
+5V (REG)	35	38	+5V (REG)
DGND	36	37	DGND



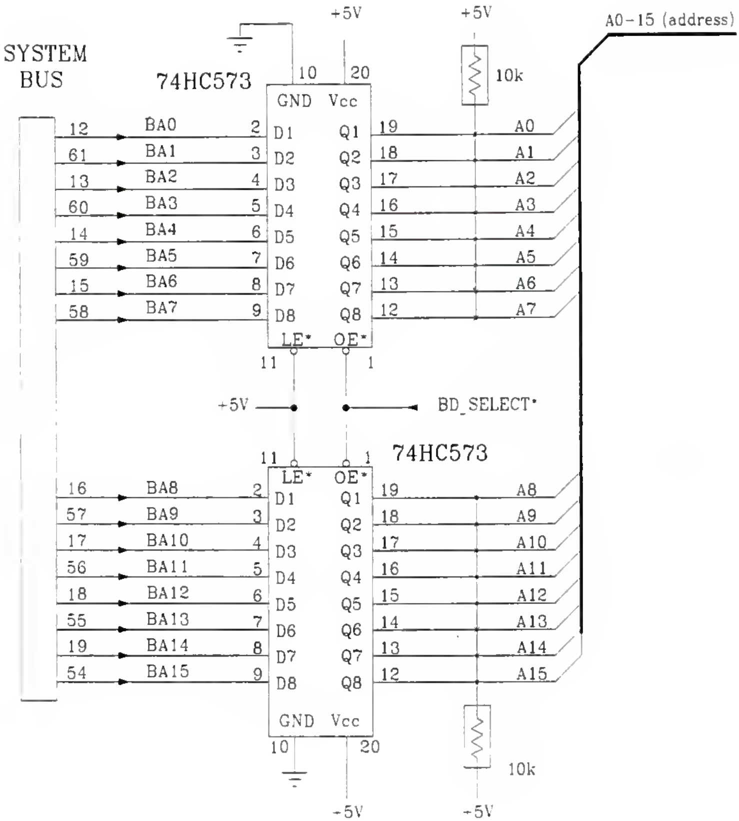
Schematic C.1 The suggested circuit for board address decoding.



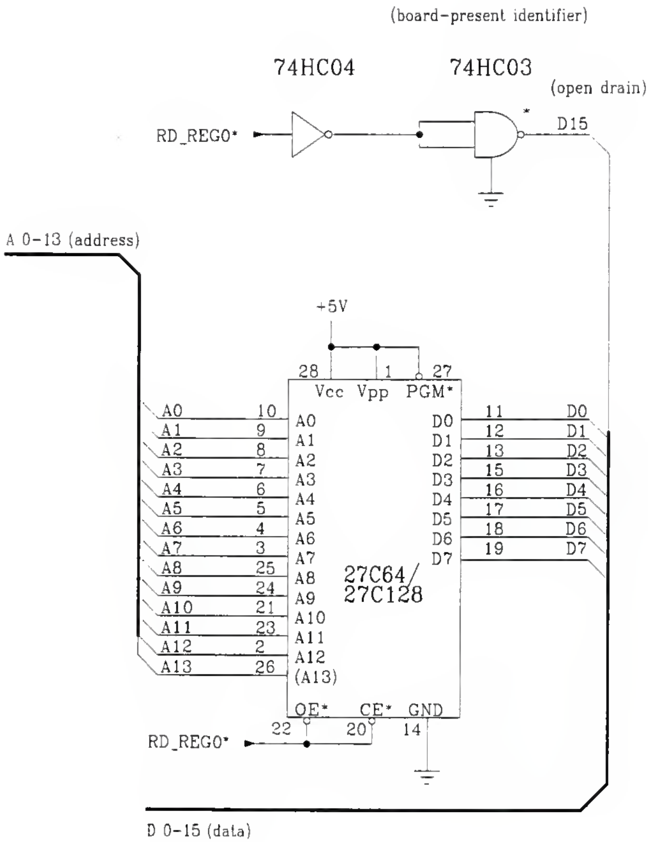
Schematic C.2 The suggested circuit for register read/write address decoding.



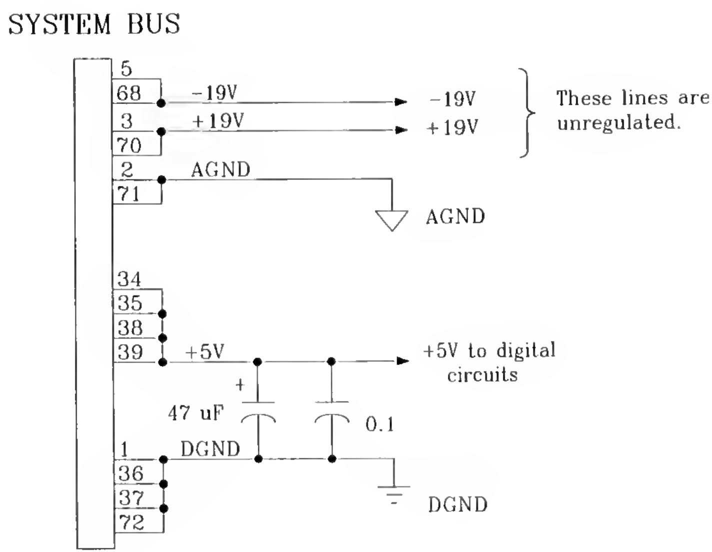
Schematic C.3 The suggested circuit for buffering the data from the bus.



Schematic C.4 The suggested circuit for buffering the 16-bit address lines from the bus.



Schematic C.5 The suggested circuit for board presence identification and board EPROM connection.



Schematic C.6 The suggested method of retrieving power from the system bus.

APPENDIX D

CIRCUIT CONSTRUCTION CONSIDERATIONS AND COMPONENT LAYOUT

The purpose of this appendix is to present details concerning the circuits that were constructed, including details about construction considerations and component layout. These circuits include the system front-end (described in Chapter 3) and the A/D board (described in Chapter 4).

D.1. Construction of the system front-end

The circuitry for the system front-end was mounted entirely on one board, as shown in Fig. D.1.1. The board selected for the system front-end was a Page Digital, Inc. P722-4, primarily because its dimensions (9 x 4.5 inches) permitted all of the components plus two edge-card connectors to be installed on it. In addition, the 36/72 edge card connection on the end of this board provided a convenient means for system bus extension.

A 50-pin connector for flat ribbon cable was used to interconnect the system controller board (the 68HC11EVB)

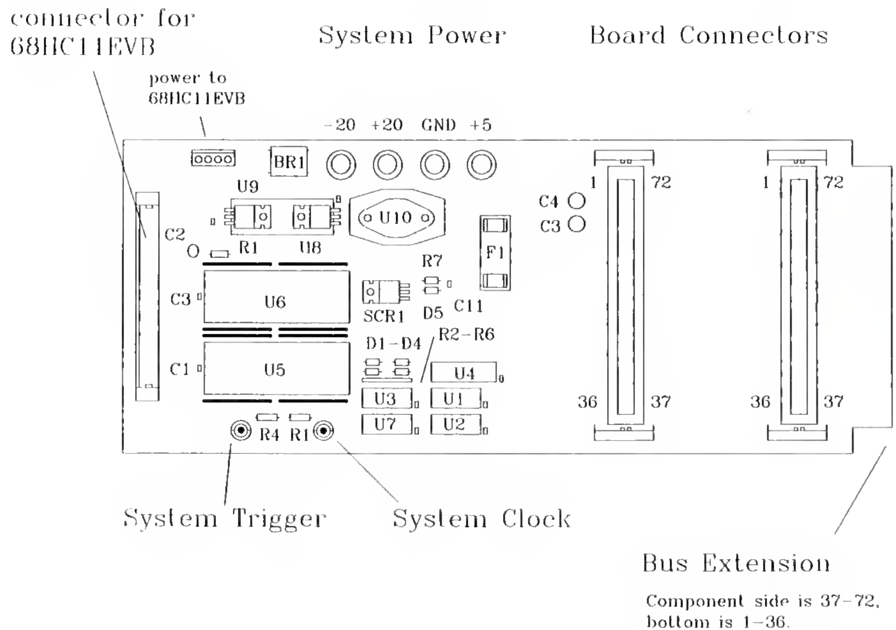


Figure D.1.1 Top view of the system front-end.

to the system front end. This connector was selected despite the fact that the connector on the 68HC11EVB has 60 pins. However, only pins between 1 and 50 are used, and the poor availability of 60 pin connectors prompted the use of the 50-pin device.

Power for the system is connected via four color coded (female) banana plugs. This method of connecting the power was selected since cables with banana terminations are readily available, and since most power supplies in the department have power outputs in the form of banana connectors.

The edge card connectors selected for the prototype were 36/72 connection (0.1" spacing), Vector type R636-1. These edge card connectors are suitable since their contacts are gold plated, and each of the contacts are split to provide a better contact with the boards. Board supports (Vector model BR27-1) were also installed to support the boards while they are installed in the system.

D.2. Construction of the A/D Board

There were many considerations made during the construction of the A/D board prototype. Among these considerations included the method of interconnecting the components: wire-wrapping was the selected for two reasons. First, this method simplifies circuit revisions, and second, printed circuit development facilities were not available (though printed circuit would have been a better alternative for the analog signals).

The number of components in the A/D board design necessitated the use of two boards. Therefore, the circuitry associated with signal handling was placed on a separate board apart from the digital control circuitry. This was advantageous since this provided a physical separation between the noisy digital signals and the sensitive analog signal components. The boards selected for the A/D board prototype were the Page Digital, Inc. model P722-4 for the digital control circuits, and the Vector model 8007 for the analog circuit. The P722-4 provided the bus connections via a 36/72 (0.1") edge card connector, and approximately 9 x 4.5 inch component mounting area. The 8007 has a ground plane and approximately 4.5 x 6 inch component mounting area. The analog board was mounted to the digital board using four, one-inch spacers.

The layout of the analog board is shown in Fig. D.2.1. The components on this board are divided into two sections: the analog signal components and digital control components. The analog signal components are all mounted above the board's ground plane. The layout of the

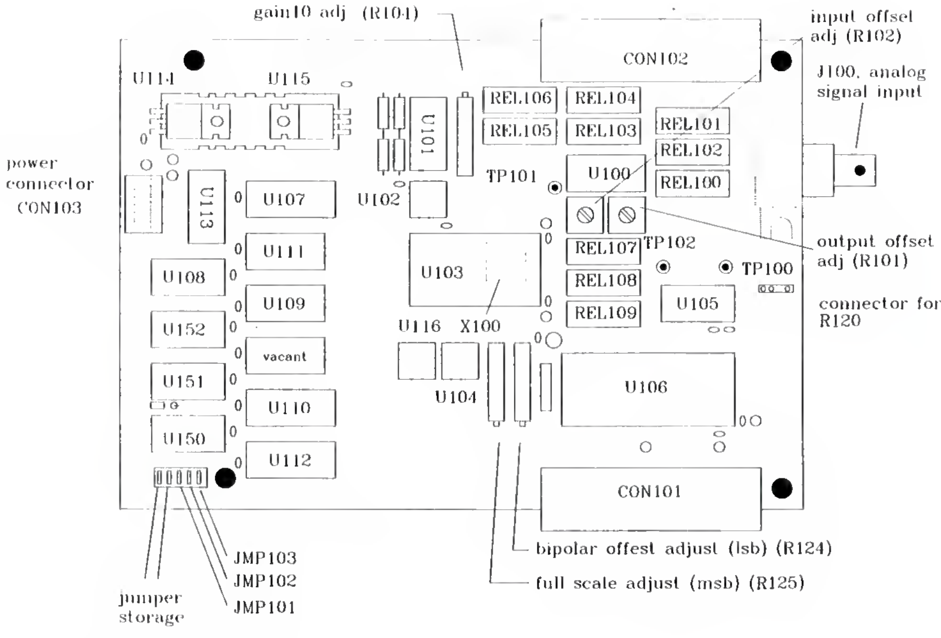


Figure D.2.1 Top view of the analog circuit board of the A/D board.

analog components were such that the length of signal interconnecting wire between the components was minimized. Therefore, the path of the analog signal can be described as a "C" shape. The signal enters through the BNC connector on the front of board, through the instrumentation amplifier (U100), over to the filter (U103), and finally over to the A/D converter (U106).

Some digital control circuitry was also placed on the analog board. These components are primarily associated with signal-relay and gain-relay control, and, therefore, not switching during a conversion sequence. This is important since noise associated with the digital transitions might be introduced to the analog circuitry.

Another group of components on the analog board provide voltage regulation for the analog signal components. These regulators are mounted on this board to minimize power line impedance for the analog signal components.

The chip sockets selected for the analog board depended upon the function of the chip. All analog signal components were provided with high-reliability (machined contact) sockets. Standard chip sockets were selected for the digital control devices.

Three connectors were used to connect the analog board to the digital control board. Two right angle 26-conductor flat-cable connectors (CON101, CON102) carry digital signals to and from the analog board. The pin assignments for these connectors is presented in Table D.2.1. Power for the analog board is received through a 5-pin connector mounted on the rear of the board (CON103).

A brass plate was used to seperate the digital board from the signal handling circuitry. This brass plate was connected to the analog board's ground plane.

The layout of the digital board is shown in Fig. D.2.2. This board has three major sections: the on-board memory, the on-board sampling clock generator, and the bus interface components. The positioning of these sections was determined after considering the location of the analog components when the analog board is mounted to the digital board. Since the on-board memory is not active during the A/D conversion process, it and its control circuitry were positioned on the right side of the board (underneath the analog circuitry). The on-board sampling clock generator is the most digitally-active circuit located on the digital board, and was therefore positioned as far from the analog circuitry as possible -- near the bus connector. The bus interface circuitry may have

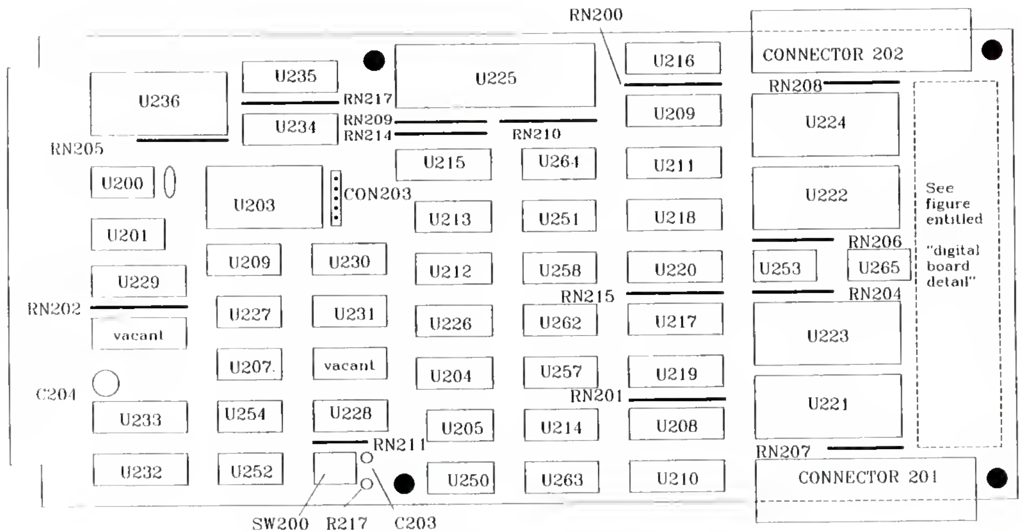


Figure D.2.2 Top view of the digital control board of the A/D board.

digital activity during the conversion process, and therefore was placed near the sampling clock. The LEDs and LED driver circuits were placed at the front of the board, as shown in Fig. D.2.3.

The socket selected for the A/D board's EPROM was a high-reliability socket since the EPROM might be removed numerous times. The position of the board's EPROM was located in a position such that it may be removed while the analog board is installed. The remainder of the sockets on the digital board are standard sockets.

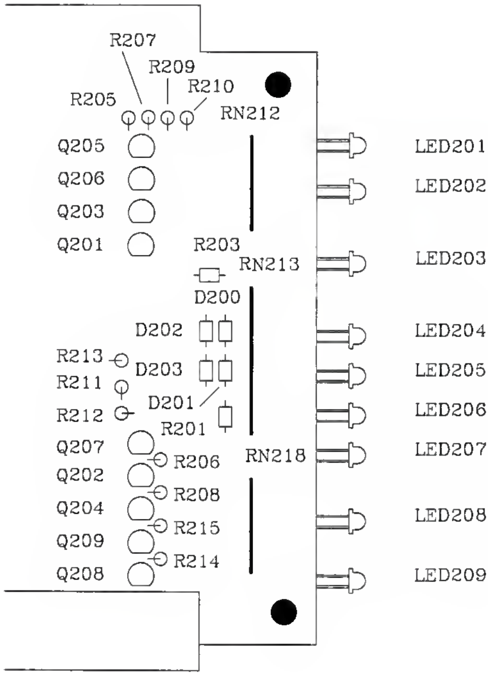


Figure D.2.3 Component placement details for the A/D digital board.

Table D.2.1 Digital board to analog board connector pin assignments.

Connector 101, 201

A/D_DAT8	1	2	A/D_DAT9
A/D_DAT10	3	4	A/D_DAT11*
A/D_DAT0	5	6	A/D_DAT1
DGND	7	8	DGND
A/D_DAT2	9	10	A/D_DAT3
A/D_DAT4	11	12	A/D_DAT5
A/D_DAT6	13	14	A/D_DAT7
DGND	15	16	DGND
D8	17	18	D9
D10	19	20	D11
D12	21	22	D13
A/D_CONV*	23	24	A/D_EOC
DGND	25	26	DGND

Connector 102, 202

DGND	1	2	DGND
A0	3	4	A1
A2	5	6	A3
A4	7	8	A5
DGND	9	10	DGND
WR_REG6*	11	12	DGND
RD_REG6*	13	14	GAIN_0
TRIG_SIG	15	16	GAIN_1
FLTR_OUT*	17	18	GAIN_2
BD_ERROR	19	20	SIG_RT_1
(reserved)	21	22	SIG_RT_3
INPUT_EN	23	24	OVERLOAD*
DGND	25	26	DGND

APPENDIX E

HARDWARE MODIFICATIONS FOR THE 68HC11EVB

The purpose of this appendix is to present the modifications necessary for the 68HC11EVB before it can be used as the system controller. The 68HC11EVB is an evaluation board for the 68HC11 single-chip microcontroller. This board has many features desireable for the system controller, and was, therefore, selected for this purpose.

However, the 68HC11EVB must be modified before it can be used as the system controller. The most efficient interface between the 68HC11EVB and the bus drivers is a direct connection to the data and address buses of the 'EVB. Unfortunately, the 68HC11EVB's data and address buses are not available via the 60-pin state connector (P1 on the 68HC11EVB). Access is gained to the address and data lines by replacing the 68HC24 port replacement unit with the jumper shown in Fig. E.1. This jumper makes the

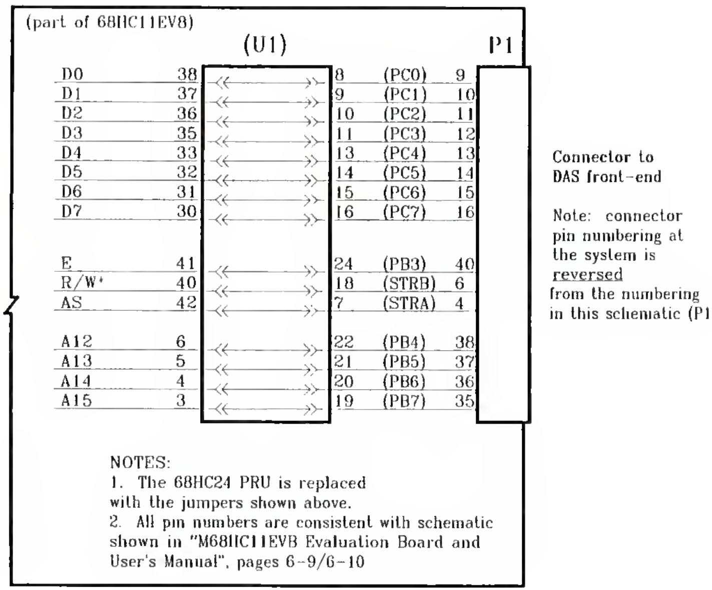


Figure E.1 The wiring modification necessary at the 68HC24 PRU socket aboard the 68HC11EVB (note: the 68HC24 is removed).

address and data lines (internal to the 68HC11EVB) available at the 60-pin state connector by way of the lines that were originally designated for the 68HC24 digital I/O. The address lines made available by this connection make the address space $A000-$B000 decodeable by the bus driver circuit (described in Chapter 3).

APPENDIX F

ALGORITHMS FOR SYSTEM TESTING AND MAINTENANCE OF PC-BASED SOFTWARE

The purpose of this appendix is to present the source code for the programs used to test the system prototype, and to provide information for source code maintenance. This C source code was compiled with the Microsoft C compiler, version 5.1, on a Zenith Z-158 PC compatible.

Software maintenance

In the event that new routines are added to the pc-based controlling software (or existing routines are modified), the following steps should be followed.

When a new function is added, the following functions must be modified (all files can be found in C:\E747\DDN):

pi_ddn.h	1) If the new function is a command, include the command in the defines for CMD_xx, where XX is the command. The values for MAX_CMD and NUM_CMDS must also be adjusted. 2) Add the function prototype information to the function definition list.
pi_disp	If the new function is a command, include a call to the function in this command-dispatcher routine.
pi_init	If the new function is a command, put logger-information pertaining to the command in the list.
pcpi_das	Add the name of the source-code file to this make-utility file.
lib_das	Add the name of the function to this list to facilitate the functions inclusion in the pcpi_das library.

Following modifications to these routines, the Microsoft Make utility should be executed. This routine updates *.obj files by compiling routines which have been altered since their last compilation. Once all routines have successfully compiled, regenerate the pcpi_das have successfully compiled, regenerate the pcpi_das library. This is done in the following manner:

1) From the DOS prompt, rename the existing pcpi_das library to PCPI_DAS.LIB to PCPI_DAS.BAK (this file is found in C:\LIB\DEVEL):

RENAME PCPI_DAS.LIB PCPI_DAS.**BAK** <enter>

2) While in the C:\E747\DDN directory, invoke the make utility with the following command:

MAKE MODEL=AS PCPI_DAS <enter>

3) Once the make utility reports no compilation errors, invoke the library generation command in the following manner:

LIB @LIB_DAS <enter>

4) Now that the pcpi_das library has been modified, create the executable file.

LINK PI_DAS <enter>

libraries: DMATH+FILE_IO+P_PLOT+PCPI_DAS <enter>

(accept defaults for each of the other prompts).

At the completion of step 4, the pcpi-das control program may be executed by typing PI_DAS <enter>.

Source-code listing

The source code is presented in the following manner. First, the headers for the files is presented, followed by system command functions. These are followed by bus driver routines, and then finally routines which emulate the commands listed in Appendix B.

PCPI-SPECIFIC ROUTINES ALTERED FOR USE WITH THE DAS

The following PCPI functions were used with the PCPI-DAS routines, however, these routines were unaltered from the original PCPI versions. Source code for these routines may be found in the PCPI literature.

int_lnln	pi_log
int_plot	pi_opin
int_sc	pi_oplog
pi_cllog	pi_opout
pi_cvt	pi_prsta
pi_date	pi_rdfil
pi_gets	pi_rdreg
pi_iface	pi_wrfil
pi_initl	pi_wrreg

PCPI-DAS MAKE UTILITY FILE

```
#   Make-file for the PCPI_DAS control program.
#
#   D.D. Nigus   4Jul89
#      (taken from:   S.A. Dyer     9Apr89)
#   Used with permission.
#
#                          Usage is:
#
#                              MAKE <model> pcpi_das
#
#                          Example for small memory-model:
#
#                              MAKE model=AS pcpi_das
#
#
opt          = /c /$(model) /FPi87 /D LINT_ARGS /W 2
d_path       = \include\devel
f_path       = \include\devel
p_path       = \include\devel
pi_path      = \include\devel
lib_path     = \lib\devel

.c.obj:
    cl $(opt) $*.c

pi_cllog.obj:   pi_cllog.c  $(pi_path)\pi_ddn.h

pi_cmd.obj:     pi_cmd.c    $(pi_path)\pi_ddn.h

pi_cvt.obj:     pi_cvt.c    $(pi_path)\pi_ddn.h

pi_date.obj:    pi_date.c   $(pi_path)\pi_ddn.h

pi_disp.obj:    pi_disp.c   $(pi_path)\pi_ddn.h

pi_gets.obj:    pi_gets.c

pi_iface.obj:   pi_iface.c  $(pi_path)\pi_ddn.h

pi_init.obj:    pi_init.c   $(pi_path)\pi_ddn.h

pi_initl.obj:   pi_initl.c  $(pi_path)\pi_ddn.h

pi_log.obj:     pi_log.c    $(pi_path)\pi_ddn.h
```

```
pi_opin.obj:    pi_opin.c   $(pi_path)\pi_ddn.h      \
                            $(f_path)\file_io.h

pi_oplog.obj:   pi_oplog.c  $(pi_path)\pi_ddn.h

pi_opout.obj:   pi_opout.c  $(pi_path)\pi_ddn.h      \
                            $(f_path)\file_io.h

pi_prsta.obj:   pi_prsta.c

pi_rdfil.obj:   pi_rdfil.c  $(pi_path)\pi_ddn.h      \
                            $(f_path)\file_io.h

pi_rdreg.obj:   pi_rdreg.c  $(pi_path)\pi_ddn.h

pi_wrfil.obj:   pi_wrfil.c  $(pi_path)\pi_ddn.h

pi_wrreg.obj:   pi_wrreg.c  $(pi_path)\pi_ddn.h

int_lnln.obj:   int_lnln.c  $(d_path)\dmath.h        \
                            $(p_path)\plot_lcl.h     \
                            $(p_path)\p_plot.h

int_plot.obj:   int_plot.c  $(p_path)\p_plot.h       \
                            $(pi_path)\pi_ddn.h

int_sc.obj:     int_sc.c    $(d_path)\dmath.h        \
                            $(p_path)\plot_lcl.h     \
                            $(p_path)\p_plot.h

#
#   ddn files . . .
#

bc_das.obj:     bc_das.c    $(d_path)\pi_ddn.h

bd_rd.obj:      bd_rd.c     $(d_path)\pi_ddn.h

bd_wr.obj:      bd_wr.c     $(d_path)\pi_ddn.h

bi_das.obj:     bi_das.c    $(d_path)\pi_ddn.h

bp_das.obj:     bp_das.c    $(d_path)\pi_ddn.h

bus_rd.obj:     bus_rd.c    $(d_path)\pi_ddn.h
```

```
bus_wr.obj:      bus_wr.c      $(d_path)\pi_ddn.h
byte_brk.obj:    byte_brk.c    $(d_path)\pi_ddn.h
cc_das.obj:      cc_das.c      $(d_path)\pi_ddn.h
cct_err.obj:     cct_err.c     $(d_path)\pi_ddn.h
cl_das.obj:      cl_das.c      $(d_path)\pi_ddn.h
ctrl_wr.obj:     ctrl_wr.c     $(d_path)\pi_ddn.h
cs_das.obj:      cs_das.c      $(d_path)\pi_ddn.h
ct_das.obj:      ct_das.c      $(d_path)\pi_ddn.h
data_dir.obj:    data_dir.c    $(d_path)\pi_ddn.h
dr_das.obj:      dr_das.c      $(d_path)\pi_ddn.h
fc_das.obj:      fc_das.c      $(d_path)\pi_ddn.h
fi_das.obj:      fi_das.c      $(d_path)\pi_ddn.h
fs_das.obj:      fs_das.c      $(d_path)\pi_ddn.h
give_val.obj:    give_val.c    $(d_path)\pi_ddn.h
gs_das.obj:      gs_das.c      $(d_path)\pi_ddn.h
gv_das.obj:      gv_das.c      $(d_path)\pi_ddn.h
help_das.obj:    help_das.c    $(d_path)\pi_ddn.h
mem_rd.obj:      mem_rd.c      $(d_path)\pi_ddn.h
mem_wr.obj:      mem_wr.c      $(d_path)\pi_ddn.h
mt_das.obj:      mt_das.c      $(d_path)\pi_ddn.h
pre_plot.obj:    pre_plot.c    $(d_path)\pi_ddn.h
rd_strob.obj:    rd_strob.c    $(d_path)\pi_ddn.h
```

PCPI-DAS MAKE UTILITY FILE

```
rs_das.obj:     rs_das.c    $(d_path)\pi_ddn.h

sc_das.obj:     sc_das.c    $(d_path)\pi_ddn.h

sing_bit.obj:   sing_bit.c  $(d_path)\pi_ddn.h

si_das.obj:     si_das.c    $(d_path)\pi_ddn.h

sr_das.obj:     sr_das.c    $(d_path)\pi_ddn.h

timer_rd.obj:   timer_rd.c  $(d_path)\pi_ddn.h

timer_wr.obj:   timer_wr.c  $(d_path)\pi_ddn.h

ts_das.obj:     ts_das.c    $(d_path)\pi_ddn.h

wr_strob.obj:   wr_strob.c  $(d_path)\pi_ddn.h

pi_das.obj:     pi_das.c    $(d_path)\pi_ddn.h
```

PCPI-DAS LIBRARY GENERATION RESPONSE FILE

```
\lib\devel\pcpi_das.lib
y
+pi_cllog+pi_cmd+pi_cvt&
+pi_date+pi_disp+pi_gets+pi_iface&
+pi_init+pi_initl+pi_log+pi_opin+pi_oplog&
+pi_opout+pi_prsta+pi_rdfil+pi_rdreg+pi_wrfil&
+int_sc+int_plot+int_lnln&
+pi_wrreg+bp_das+cc_das+cs_das+ct_das+dr_das&
+fi_das+fs_das+gs_das+gv_das+mt_das+bi_das&
+rs_das+sc_das+si_das+sr_das+ts_das+bus_wr&
+bc_das+cl_das+fc_das+pre_plot+help_das&
+mem_wr+mem_rd+timer_rd+bus_rd+cct_err&
+wr_strob+rd_strob+data_dir+ctrl_wr+sing_bit&
+timer_wr+bd_rd+bd_wr+give_val+byte_brk
pcpi_das.lst
```

```
/************************************************************
 *
 *  SOURCE FILE:     pi_ddn.h
 *
 *
 *  DESCRIPTION:     Header file for PC peripheral
 *                   interface (PCPI) control program.
 *                   This is a modified version of
 *                   Stephen A. Dyer's "pi.h".
 *                   Used with permission.
 *
 *
 *  DOCUMENTATION
 *  FILES:           None.
 *
 *
 *  AUTHOR:          Copyright 1989
 *                   Durwin D. Nigus
 *
 *
 *  DATE CREATED:    24 July 1989       Version 1.00
 *
 *
 *  REVISIONS:       None.
 *
 *
************************************************************/

#define FALSE         0
#define TRUE          1

#define LO            0
#define HI            1

#define DISABLE       0
#define ENABLE        1

#define INACTIVE      0
#define ACTIVE        1

#define OFF           0
#define ON            1

#define NO            0
#define YES           1
```

```
#define EPS           1.0E-3

#define BUF_LEN          81       /* Maximum length of     */
                                  /* character buffers.    */

#define PRMPT_MAX         25      /* Maximum number of     */
                                  /* characters in the     */
                                  /* prompt buffer.        */

#define MAX_DATA       8192       /* Maximum length of     */
                                  /* buffer for digitized  */
                                  /* data.                 */

#define NUM_REGS          8       /* Number of read/write  */
                                  /* registers on the      */
                                  /* PCPI external module. */

#define NUM_RDREGS        8       /* Number of read        */
                                  /* registers on the      */
                                  /* PCPI.                 */

#define NUM_WRREGS       16       /* Number of write       */
                                  /* registers on the      */
                                  /* PCPI.                 */

#define MAX_BDS          16       /* Number of boards      */
                                  /* allowed in the DAS.   */


/*-------------------------------------------------------*/
/*  Errors.                                               */
/*-------------------------------------------------------*/
#ifndef NORMAL
#define NORMAL            0       /* Normal return.        */
#endif

#define ERR_INITBP       10       /* pi_init():  base port */
                                  /* not found in the      */
                                  /* environment.          */

#define ERR_INITTST      11       /* pi_init():  TEST_DAS  */
                                  /* not found in the      */
                                  /* environment.          */
```

```
#define ERR_INITXTL     12          /* pi_init():  A/D board */
                                    /* xtal freq not found   */
                                    /* in the environment.   */

#define ERR_OPEN        20          /* pi_oplog():           */
                                    /* pi_opin ():           */
                                    /* pi_opout():           */
                                    /* cannot open file      */

#define ERR_OPENACC     21          /* pi_oplog():  log file */
                                    /* having the given      */
                                    /* filename already      */
                                    /* exists.               */
                                    /* pi_opin ():  file     */
                                    /* does not exist.       */
                                    /* pi_opout():  file     */
                                    /* already exists.       */

#define ERR_RDFIL       30          /* pi_rdfil(): could not */
                                    /* complete reading      */
                                    /* the requested input-  */
                                    /* data file.            */

#define ERR     NORMAL + 1          /* Abnormal return flag. */


/*------------------------------------------------------*/
/*  Register definitions.                               */
/*------------------------------------------------------*/
#define RGE           8             /* Control port for      */
                                    /* -BDINIT, -MRESET.     */
#define PI_IFACE     12             /* Control port for PCPI */
                                    /* to external-module    */
                                    /* three-state buffers.  */


/*======================================================*/
/*  Command codes.                                      */
/*------------------------------------------------------*/
#define MAX_CMDS     45             /* Maximum number of     */
                                    /* commands allowed.     */
#define NUM_CMDS     40             /* Number of commands    */
                                    /* (including Error).    */
```

```
                                        /* Command codes.           */
#define CMD_ERROR    0                  /* Error:  invalid          */
                                        /*         command.         */
#define CMD_R        1                  /* Read from register.      */
#define CMD_W        2                  /* Write to register.       */

#define CMD_SEMI     3                  /* Comment.                 */

#define CMD_E        4                  /* Enable interface.        */
#define CMD_D        5                  /* Disable interface.       */

#define CMD_I        6                  /* Toggle initialization */
#define CMD_IB       7                  /* Activate -BDINIT.        */
#define CMD_IM       8                  /* Activate -MRESET.        */
#define CMD_IA       9                  /* Activate all inits.      */
#define CMD_ID      10                  /* Deactivate all inits. */

#define CMD_L       11                  /* Toggle session-logger */

#define CMD_QUIT    12                  /* Quit the session.        */

#define CMD_P       13                  /* Plot contents of data */
                                        /* buffer on display.       */
#define CMD_PP      14                  /* Plot contents of data */
                                        /* buffer on pen plotter */

#define CMD_SI      15                  /* System front-end         */
                                        /* initialization.          */
#define CMD_BI      16                  /* Board initialization. */

#define CMD_CT      17                  /* Configure system         */
                                        /* trigger.                 */
#define CMD_CC      18                  /* Configure system         */
                                        /* clock.                   */
#define CMD_BP      19                  /*  Determine boards        */
                                        /*  present.                */

#define CMD_TS      20                  /*  Trigger selection       */
                                        /*  for A/D board.          */
#define CMD_CS      21                  /*  Clock select for        */
                                        /*  A/D board.              */
#define CMD_FS      22                  /*  Full-scale signal       */
                                        /*  level adjust for A/D */
                                        /*  board.                  */
#define CMD_FI      23                  /*  Filter enable/disable*/
```

```
                                        /*  for the A/D board.    */
#define CMD_SR      24                  /*  Set sample rate       */
                                        /*  on A/D board.         */
#define CMD_BC      25                  /*  Begin conversion on   */
                                        /*  A/D board.            */
#define CMD_SC      26                  /*  Stop conversion.      */

#define CMD_GS      27                  /*  Get A/D board status.*/

#define CMD_SB      28                  /*  Set active board      */
                                        /*  address.              */
#define CMD_DR      29                  /*  Display contents of   */
                                        /*  bus drivers.          */

#define CMD_TR      30                  /*  Trace mode toggle.    */

#define CMD_STEP    31                  /*  I/O-step toggle.      */

#define CMD_RS      32                  /*  Retrieve samples.     */

#define CMD_MT      33                  /*  On-board memory       */
                                        /*  test.                 */

#define CMD_GV      34                  /*  Get converter value.  */

#define CMD_FC      35                  /*  Filter configure.     */

#define CMD_CL      36                  /*  Calibration routine   */
                                        /*  for A/D board.        */

#define CMD_BR      37                  /*  Board write command.  */
#define CMD_BW      38                  /*  Board read command.   */

#define CMD_HELP    39                  /*  Display key-help.     */

/*=====================================================*/
/*  Type-int parameter table.                          */
/*-----------------------------------------------------*/
#define REG               1     /* Register most recently  */
                                /* read from or written    */
                                /* to.                     */

#define VALUE             2     /* Value most recently     */
                                /* used by a register-read */
```

```
                                /* or write.                 */

#define IFACE               3   /* State of PCPI three-      */
                                /* state buffers.            */

#define BDINIT              4   /* State of -BDINIT line.    */

#define MRESET              5   /* State of -MRESET line.    */

#define SCREEN              6   /* State of screen-logger.   */

#define LOG                 7   /* State of session-         */
                                /* logger.                   */

#define LOG_NO              8   /* Number of current log     */
                                /* entry.                    */

#define LAST_LOG            9   /* Number of entry last      */
                                /* made to session-logger.   */

#define QUIT               10   /* State of request to       */
                                /* terminate the present     */
                                /* session.                  */

#define DATA_LEN           11   /* Length of valid data in   */
                                /* data buffer.              */

#define LEN_INFILE         12   /* Length of valid data in   */
                                /* input-data file.          */

#define LEN_OUTFILE        13   /* Length of valid data in   */
                                /* last-written output-      */
                                /* data file.                */

#define FA_LC              14   /* Loop constant for A/D     */
                                /* conversion.               */

#define FD_LC              15   /* Loop constant for D/A     */
                                /* conversion.               */
```

```
/*=========================================================*/
/*  PCPI information and command structures.               */
/*---------------------------------------------------------*/
#define     L_LFN       30    /* Max. length of filename */
                              /* for session-log.        */
#define     L_LDATE     25    /* Length of date-string   */
                              /* for session-log.        */

#define     L_INFN      30    /* Maximum length of file- */
                              /* name for input-         */
                              /* data file.              */
#define     L_OUTFN     30    /* Maximum length of file- */
                             /* name for output-         */
                              /* data file.              */

typedef struct  pcpi_table
    {
    double      flt[21];      /* Type-double parameter   */
                              /* table.                  */
    int         fix[21];      /* Type-int parameter      */
                              /* table.                  */
    int         inreg[16];    /* Values contained in bus */
                              /* driver input-registers  */
                              /* (regs. written to by    */
                              /* computer/interface).    */
    int         outreg[8];    /* Values contained in bus */
                              /* driver output-registers */
                              /* (regs. read from by the */
                              /* computer/interface).    */
    double      con_d[11];    /* Type-double conversion  */
                              /* constants.              */
    int         con_i[11];    /* Type-int conversion     */
                              /* constants.              */
    unsigned    base_port;    /* Base port for register  */
                              /* address space.          */
    FILE        *logger;      /* Handle for session-     */
                              /* logger file.            */
    char        log_fn[L_LFN + 1];
                              /* Filename for session-   */
                              /* logger.                 */
    char        log_date[L_LDATE + 1];
                              /* Date for session-log.   */
    FILE        *infile;      /* Handle for input-data   */
                              /* file.                   */
    FILE        *outfile;     /* Handle for output-data  */
```

```
                               /* file.                  */
    char        in_file[L_INFN + 1];
                               /* Filename for input-data */
                               /* file.                   */
    char        out_file[L_OUTFN + 1];
                               /* Filename for output-    */
                               /* data file.              */
    } PCPI_TABLE;


typedef struct pcpi_command
    {
    char    cmd_str[5];        /* Keyboard-entry string   */
                               /* for the command.        */
    char    log_format[80];    /* Format-string to be     */
                               /* used by the session     */
                               /* logger.                 */
    } PCPI_COMMAND;


/*=========================================================*/
/*  Definitions and declarations of static variables.      */
/*---------------------------------------------------------*/
#ifndef PI_INIT
    extern  PCPI_TABLE      pi;
    extern  PCPI_COMMAND    commands[MAX_CMDS];
    extern  char            pi_buf1[BUF_LEN];
    extern  char            pi_buf2[BUF_LEN];
    extern  char            pi_logbuf[BUF_LEN];
    extern  char            pi_tempbuf[BUF_LEN];
    extern  int             pi_data[MAX_DATA];

    extern  int             plot_len;       /*-----------*/
    extern  long            xtal_freq;
    extern  int             initialized[MAX_BDS];
    extern  int             trace;          /* ddn       */
    extern  int             stepper;        /* additions */
    extern  unsigned        num_pre_trig;   /*           */
    extern  unsigned        num_post_trig;  /*           */

    extern  int             test_das;       /* if YES,   */
                                            /* no bus    */
                                            /* i/o.      */
                                            /*-----------*/
```

```
#else
    PCPI_TABLE        pi;
    PCPI_COMMAND      commands[MAX_CMDS];
    char              pi_buf1[BUF_LEN];
    char              pi_buf2[BUF_LEN];
    char              pi_logbuf[BUF_LEN];
    char              pi_tempbuf[BUF_LEN];
    int               pi_data[MAX_DATA];
    int               plot_len;
    long              xtal_freq;
    int               initialized[MAX_BDS];
    int               trace;
    int               stepper;
    unsigned          num_pre_trig;
    unsigned          num_post_trig;
    int               test_das;
#endif


/*=========================================================*/
/*  Function declarations.                                 */
/*---------------------------------------------------------*/
#ifdef  LINT_ARGS

int     inp       (unsigned);
int     outp      (unsigned, int);


int     int_lnln(int, int, int, int, int, double, double,
                double, int[], int, double, double,
                double, int, char);
int     int_plot(int, int, int, int, int[], char[],
                char[], char[], char[], char[], int,
                double);
int     int_sc   (int[], int, int, double, int, double,
                double *, double *, double *);


int     pi_cllog (void);
int     pi_cmd   (char *);
int     pi_cvt   (char *);

int     pi_date  (void);
int     pi_disp  (int, int *);
int     pi_gets  (int, char *);
```

```
int      pi_iface (int);
int      pi_init  (void);
int      pi_initl (int, int);
int      pi_log   (char *);
int      pi_opin  (char *);

int      pi_oplog (char *);
int      pi_opout (char *);
char     *pi_prsta(char *, char *, int);
int      pi_rdfil (void);
int      pi_rdreg (int);

int      pi_wrfil (void);
int      pi_wrreg (int, int);

/*-----------------------------------------------------*/
/*   PCPI_DAS-specific functions.                      */
/*-----------------------------------------------------*/

int        mem_wr (int, unsigned, unsigned, int, int);
unsigned   mem_rd (int, unsigned, int, int);
int        bus_rd (int);
int        bus_wr (int, int);

int        wr_strob (void);
int        rd_strob (int, int);
int        data_dir (int, int, int);
int        ctrl_wr (int, int, int);

int        sing_bit (int, int, int);
int        timer_wr (int, int, unsigned);
unsigned   timer_rd (int, int);
unsigned   bd_rd (int, int);

int        pre_plot (void);
int        bd_wr (int, int, unsigned, int);
int        give_val (int, int);
int        byte_brk (unsigned, int *, int *);
int        bp_das (void);

int        cc_das (void);
int        cs_das (int);
int        ct_das (void);
int        dr_das (void);
```

```
int        fi_das (int);

int        fs_das (int);
int        gs_das (int);
int        gv_das (int);
int        mt_das (int);
int        bi_das (int);

int        rs_das (int);
int        sc_das (int);
int        si_das (void);
int        sr_das (int);

int        ts_das (int);
int        bc_das (int);
int        cl_das (int);
int        fc_das (int);

int        help_das (void);

#else

int      inp      ();
int      outp     ();

int      int_lnln ();
int      int_plot ();
int      int_sc   ();

int      pi_cllog ();
int      pi_cmd   ();
int      pi_cvt   ();

int      pi_date  ();
int      pi_disp  ();
int      pi_gets  ();

int      pi_iface ();
int      pi_init  ();
int      pi_initl ();
int      pi_log   ();
int      pi_opin  ();
```

```
int       pi_oplog ();
int       pi_opout ();
char      *pi_prsta();
int       pi_rdfil ();
int       pi_rdreg ();

int       pi_wrfil ();
int       pi_wrreg ();

/*-------------------------------------------------------*/
/*  PCPI_DAS-specific function declarations.             */
/*-------------------------------------------------------*/

int         mem_wr ();
unsigned    mem_rd ();
unsigned    timer_rd ();
int         bus_rd ();
int         bus_wr ();

int         pre_plot ();
int         wr_strob ();
int         rd_strob ();
int         data_dir ();
int         ctrl_wr ();

int         sing_bit ();
int         ctrl_bd ();
int         timer_wr ();
unsigned    bd_rd ();

int         bd_wr ();
int         give_val ();
int         byte_brk ();
int         bp_das ();

int         cc_das ();
int         cs_das ();
int         ct_das ();
int         dr_das ();
int         fi_das ();
```

```
int         fs_das ();
int         gs_das ();
int         gv_das ();
int         mt_das ();
int         bi_das ();

int         rs_das ();
int         sc_das ();
int         si_das ();
int         sr_das ();

int         ts_das ();
int         bc_das ();
int         cl_das ();
int         fc_das ();

int         help_das ();

#endif


/***********************************************************/
/*  Material specific to DAS routines.                     */
/***********************************************************/

/*=======================================================*/
/*  Bus-driver (PPI) information                          */
/*-------------------------------------------------------*/

/*-------------------------------------------------------*/
/*  Bus-driver register names and addresses.              */
/*-------------------------------------------------------*/
#define    PPI1_CTRL        3
#define    PPI2_CTRL        7
#define    DAT_CTL_REG      PPI1_CTRL

#define    REG_ADR          2
#define    BUS_CTRL         4

#define    DAT_LSB          1
#define    DAT_MSB          0

#define    ADDR_LSB         6
#define    ADDR_MSB         5
```

```
/*-------------------------------------------------------------*/
/*  PPI configuration-register values (for PPI1_CTRL          */
/*  and PPI2_CTRL registers).                                  */
/*-------------------------------------------------------------*/
#define    BUS_READ     0x92
#define    BUS_WRITE    0x80
#define    BUS_ADDR     0x80


/*-------------------------------------------------------------*/
/*  Define the multipliers for board-number and               */
/*  register-number within REG_ADR.                            */
/*                                                             */
/*  The multiplier for the top 4-bits is 0x10.                 */
/*-------------------------------------------------------------*/
#define    REG_MULT     0x10
#define    BD_MULT      0x01


/*=============================================================*/
/*  define the bus-control bits                                */
/*-------------------------------------------------------------*/
#define    ACT_POSITION   16    /*  bit 4 of the id is     */
                                /*  "active" identifier    */
                                /*  active -low or -high   */
#define    ACT_LO         0
#define    ACT_HI         1
```

```
/*-----------------------------------------------------*/
/*   Information for the bus-control lines.            */
/*   Each of these one-byte IDs contain information    */
/*   about each bus-control line:                      */
/*
/*         bits 0 - 3    is the bit position;          */
/*         bit  4        identifies whether the bit    */
/*                       is active low or high (see    */
/*                       above);                       */
/*         bits 5 - 7    unused.                       */
/*-----------------------------------------------------*/

#define   BUS_WR_ID       0  + ACT_LO * ACT_POSITION
#define   BUS_RD_ID       1  + ACT_LO * ACT_POSITION
#define   SYS_X_TRIG_ID   4  + ACT_HI * ACT_POSITION
#define   SYS_TR_LO_ID    5  + ACT_LO * ACT_POSITION
#define   SYS_X_CLK_ID    6  + ACT_HI * ACT_POSITION

/*=====================================================*/
/*  A/D board information                              */
/*-----------------------------------------------------*/

/*-----------------------------------------------------*/
/*  Register information.                              */
/*  The register address is in bits 0-3, and           */
/*  bits 4, 5, and 6 identify the byte(s) from which   */
/*  the register is addressable.                       */
/*                                                     */
/*  The data mask is used to determine the appropriate */
/*  I/O register.                                      */
/*-----------------------------------------------------*/

#define  DATA_LOW      16    /*  bit 4 means LSB       */
#define  DATA_HIGH     32    /*  bit 5 means MSB       */
#define  DATA_BOTH     64    /*  bit 6 means 16-bit    */

#define  DATA_MASK     DATA_LOW + DATA_HIGH + DATA_BOTH
```

```
#define  EPROM          0     + DATA_LOW
#define  BD_PRESENT     0     + DATA_HIGH
#define  ON_BD_MEM      1     + DATA_BOTH
#define  TIMER          2     + DATA_HIGH
#define  STATUS         3     + DATA_LOW
#define  CTRL1          4     + DATA_LOW
#define  TRIG_ADDR      5     + DATA_BOTH
#define  FILTER         6     + DATA_HIGH
#define  AD_DIRECT      7     + DATA_BOTH
#define  CNT_STAT_CON   0x0B  + DATA_LOW
#define  CTRL2          0x0C  + DATA_LOW

/*-------------------------------------------------------*/
/*  control_status register information                  */
/*-------------------------------------------------------*/

#define   CON_STAT_CONF  0x82 /* configuration byte for */
                              /* the con_stat chip is   */
                              /* (1000 0010 b)          */

/*=======================================================*/
/*  Control and status register data_id specifications   */
/*  and definitions.                                     */
/*                                                       */
/*  The definition of data_id is as follows:             */
/*                                                       */
/*       bit0: \   0= reg1, 1 = reg2, 2 = status         */
/*       bit1: /                                         */
/*       bit2:  \                                        */
/*       bit3:   > bit start value (0-7)                 */
/*       bit4:  /                                        */
/*       bit5:  \                                        */
/*       bit6:   > number of bits (0-7)                  */
/*       bit7:  /                                        */
/*                                                       */
/*    where bit0 is the least-sig-bit of data_id         */
/*                                                       */
/*    Data_id is used to identify a particular set of    */
/*    bits: on which register they are located,          */
/*    where within the register they are located,        */
/*    and how many bits make up the region.              */
/*                                                       */
/*-------------------------------------------------------*/
```

```
#define  BIT_POS        4
#define  NUM_BTS        32
#define  CONTROL1       0
#define  CONTROL2       1
#define  STS            2

/*-------------------------------------------------------*/
/*  Region definitions.                                  */
/*-------------------------------------------------------*/

#define  BD_MODE        CONTROL1 + 0*BIT_POS + 2*NUM_BTS
#define  TRIG_SEL       CONTROL1 + 2*BIT_POS + 2*NUM_BTS
#define  TRIG_EDGE      CONTROL1 + 4*BIT_POS + 1*NUM_BTS
#define  CLOCK_SEL      CONTROL1 + 5*BIT_POS + 2*NUM_BTS
#define  FIL_EN         CONTROL1 + 7*BIT_POS + 1*NUM_BTS

#define  SIG_SEL        CONTROL2 + 0*BIT_POS + 2*NUM_BTS
#define  GAIN_SEL       CONTROL2 + 2*BIT_POS + 3*NUM_BTS
#define  TMR_CNT        CONTROL2 + 5*BIT_POS + 2*NUM_BTS
#define  PROTECT        CONTROL2 + 7*BIT_POS + 1*NUM_BTS

#define  BD_ER          STS + 0*BIT_POS + 1*NUM_BTS
#define  SAMP_SER_END   STS + 1*BIT_POS + 1*NUM_BTS
#define  TRIG_RECVD     STS + 2*BIT_POS + 1*NUM_BTS
#define  PTR_WRAP       STS + 3*BIT_POS + 1*NUM_BTS
#define  EOC            STS + 4*BIT_POS + 1*NUM_BTS

/*-------------------------------------------------------*/
/*   Bit-field information for the status register.      */
/*-------------------------------------------------------*/

#define   BD_ER_ACT      ACT_LO    /*  active values for  */
#define   SAMP_END_ACT   ACT_LO    /*  status bits        */
#define   TR_REC_ACT     ACT_LO    /*                     */
#define   PTR_WR_ACT     ACT_LO    /*                     */
#define   EOC_ACT        ACT_HI    /*                     */
```

```
/*-------------------------------------------------------*/
/*  Control values for each specific region.             */
/*-------------------------------------------------------*/

#define    STBY          0    /*   BD_MODE                 */
#define    CONV_NOW      1    /*   Board mode.             */
#define    CONV_TR       2    /*                           */
#define    CONV_TR_PRE   3    /*                           */

#define    TR_SIG        0    /*   TRIG_SEL                */
#define    TR_SIG_BUS    1    /*   Trigger selection.      */
#define    TR_BUS        2    /*                           */
#define    TR_PANEL      3    /*                           */

#define    TR_EDGE_POS   0    /*   TRIG_EDGE               */
#define    TR_EDGE_NEG   1    /*   Trigger edge select.    */

#define    CLK_INT_HI    0    /*   CLOCK_SEL               */
#define    CLK_INT_LO    1    /*   Clock selection.        */
#define    CLK_BUS       2    /*                           */
#define    CLK_PANEL     3    /*                           */

#define    FILTER_OUT    0    /*   FIL_EN                  */
#define    FILTER_IN     1    /*   Filter control.         */

#define    SIG_REMOVE    0    /*   SIG_SEL                 */
#define    SIG_APPLIED   1    /*   Signal source select.   */
#define    SIG_REF       2    /*                           */
#define    SIG_TR_LVL    3    /*                           */

#define    GAIN_1        0    /*   GAIN_SEL                */
#define    GAIN_10       1    /*   Gain selection.         */
#define    GAIN_100      2    /*                           */
#define    GAIN_200      3    /*                           */
#define    GAIN_500      4    /*                           */

#define    TMR_SAMP_PD   0    /*   TMR_CNT                 */
#define    TMR_ONESHT    1    /*   Timer control register  */
#define    TMR_SAMP_CNT  2    /*   addressing controls.    */
#define    TMR_CTL_REG   3    /*                           */

#define    PROTECT_ON    0    /*   PROTECT                 */
#define    PROTECT_OFF   1    /*   Signal input overloard  */
                              /*   protection.             */
```

```
/*-------------------------------------------------------*/
/*      One-shot delay time adjustment.                  */
/*   This delay is used to allow the sample-and-hold     */
/*   amplifier to settle.  Time of the delay is given    */
/*   by                                                  */
/*              t_d = DLY_1SHT / XTAL_FREQ               */
/*                                                       */
/*   where t_d is the one-shot delay time, in seconds;   */
/*   DLY_1SHT is the 16-bit one-shot counter value, and  */
/*   XTAL_FREQ is the frequency of the on-board          */
/*   oscillator's crystal, in Hz.                        */
/*-------------------------------------------------------*/

#define   DLY_1SHT    4

/*-------------------------------------------------------*/
/*    A/D board sampling frequency bounds using the      */
/*    internal oscillator.                               */
/*-------------------------------------------------------*/
#define   F_LOW       XTAL_FREQ / 1024.0 / 65536.0
                                 /*  This is the       */
                                 /*  slowest sampling  */
                                 /*  freq for the A/D  */
                                 /*  board.            */

#define   F_HIGH      150000.0   /*  150 kHz is the    */
                                 /*  maximum conver-   */
                                 /*  sion rate for     */
                                 /*  A/D board.        */

#define   F_MID       100.0      /*  freq of division  */
                                 /*  between int.      */
                                 /*  oscillators.      */

/*-------------------------------------------------------*/
/*    controls for on-board filter                       */
/*-------------------------------------------------------*/
#define   FIL_XTL        2.4576e6 /*  Filter XTAL freq */
                                  /*  (in Hz).         */
#define   FIL_CDC        0x1e     /*  Clock divide     */
                                  /*  code address.    */
```

```
/*------------------------------------------------------------*/
/*  Values for memory writing - quick or normal control */
/*------------------------------------------------------------*/
#define   FAST         10
#define   REGULAR      FAST + 3

/*------------------------------------------------------------*/
/*  on-board memory base address                           */
/*------------------------------------------------------------*/
#define   FIRST_SAMP  1
```

```
/**************************************************************
 *
 *   SOURCE FILE:     pi_mod.h
 *
 *
 *   DESCRIPTION:     Header file for PC peripheral
 *                    interface (PCPI) control program.
 *                    This is a modified version of
 *                    Stephen A. Dyer's "pi.h".
 *                    Used with permission.
 *
 *
 *   DOCUMENTATION
 *   FILES:           None.
 *
 *
 *   AUTHOR:          Copyright 1989
 *                    Durwin D. Nigus
 *
 *
 *   DATE CREATED:    12 August 1989    Version 1.00
 *
 *
 *   REVISIONS:       None.
 *
 *
 **************************************************************/

#define FALSE           0
#define TRUE            1

#define LO              0
#define HI              1

#define DISABLE         0
#define ENABLE          1

#define INACTIVE        0
#define ACTIVE          1

#define OFF             0
#define ON              1

#define NO              0
#define YES             1
```

```
#define EPS           1.0E-3

#define BUF_LEN          81      /* Maximum length of    */
                                 /* character buffers.   */

#define MAX_DATA       8192      /* Maximum length of    */
                                 /* buffer for digitized */
                                 /* data.                */

#define NUM_REGS          8      /* Number of read/write */
                                 /* registers on the     */
                                 /* PCPI external module. */


#define NUM_RDREGS        8      /* Number of read       */
                                 /* registers on the     */
                                 /* PCPI.                */

#define NUM_WRREGS       16      /* Number of write      */
                                 /* registers on the     */
                                 /* PCPI.                */

#define MAX_BDS          16      /* Number of boards     */
                                 /* allowed in the DAS.  */


/*----------------------------------------------------*/
/*  Errors.                                           */
/*----------------------------------------------------*/
#ifndef NORMAL
#define NORMAL            0      /* Normal return.       */
#endif

#define ERR_INITBP       10      /* pi_init():  base port */
                                 /* not found in the     */
                                 /* environment.         */

#define ERR_INITTST      11      /* pi_init():  TEST_DAS  */
                                 /* not found in the     */
                                 /* environment.         */
```

```
#define ERR_OPEN        20          /* pi_oplog():            */
                                    /* pi_opin ():            */
                                    /* pi_opout():            */
                                    /* cannot open file       */

#define ERR_OPENACC     21          /* pi_oplog():  log file */
                                    /* having the given       */
                                    /* filename already       */
                                    /* exists.                */
                                    /* pi_opin ():  file      */
                                    /* does not exist.        */
                                    /* pi_opout():  file      */
                                    /* already exists.        */

#define ERR_RDFIL       30          /* pi_rdfil(): could not */
                                    /* complete reading       */
                                    /* the requested input-   */
                                    /* data file.             */


/*-------------------------------------------------------*/
/*  Register definitions.                                 */
/*-------------------------------------------------------*/
#define RGE            8            /* Control port for       */
                                    /* -BDINIT, -MRESET.      */
#define PI_IFACE      12            /* Control port for PCPI */
                                    /* to external-module     */
                                    /* three-state buffers.   */


/*=======================================================*/
/*  Command codes.                                        */
/*-------------------------------------------------------*/
#define MAX_CMDS      45            /* Maximum number of      */
                                    /* commands allowed.      */
#define NUM_CMDS      40            /* Number of commands     */
                                    /* (including Error).     */
```

```
                                        /* Command codes.          */
#define CMD_ERROR    0                  /* Error:  invalid         */
                                        /*         command.        */
#define CMD_R        1                  /* Read from register.     */
#define CMD_W        2                  /* Write to register.      */

#define CMD_SEMI     3                  /* Comment.                */

#define CMD_E        4                  /* Enable interface.       */
#define CMD_D        5                  /* Disable interface.      */

#define CMD_I        6                  /* Toggle initialization */
#define CMD_IB       7                  /* Activate -BDINIT.       */
#define CMD_IM       8                  /* Activate -MRESET.       */
#define CMD_IA       9                  /* Activate all inits.     */
#define CMD_ID      10                  /* Deactivate all inits. */

#define CMD_L       11                  /* Toggle session-logger */

#define CMD_QUIT    12                  /* Quit the session.       */

#define CMD_P       13                  /* Plot contents of data */
                                        /* buffer on display.      */
#define CMD_PP      14                  /* Plot contents of data */
                                        /* buffer on pen plotter */

#define CMD_SI      15                  /* System front-end        */
                                        /* initialization.         */
#define CMD_BI      16                  /* Board initialization. */

#define CMD_CT      17                  /* Configure system        */
                                        /* trigger.                */
#define CMD_CC      18                  /* Configure system        */
                                        /* clock.                  */
#define CMD_BP      19                  /*  Determine boards       */
                                        /*  present.               */

#define CMD_TS      20                  /*  Trigger selection      */
                                        /*  for A/D board.         */
#define CMD_CS      21                  /*  Clock select for       */
                                        /*  A/D board.             */
#define CMD_FS      22                  /*  Full-scale signal      */
                                        /*  level adjust for A/D */
                                        /*  board.                 */
#define CMD_FI      23                  /*  Filter enable/disable*/
```

```
                                  /*  for the A/D board.    */
#define CMD_SR      24            /*  Set sample rate       */
                                  /*  on A/D board.         */
#define CMD_BC      25            /*  Begin conversion on   */
                                  /*  A/D board.            */
#define CMD_SC      26            /*  Stop conversion.      */

#define CMD_GS      27            /*  Get A/D board status.*/

#define CMD_SB      28            /*  Set active board      */
                                  /*  address.              */
#define CMD_DR      29            /*  Display contents of   */
                                  /*  bus drivers.          */

#define CMD_TR      30            /*  Trace mode toggle.    */

#define CMD_STEP    31            /*  I/O-step toggle.      */

#define CMD_RS      32            /*  Retrieve samples.     */

#define CMD_MT      33            /*  On-board memory       */
                                  /*  test.                 */

#define CMD_GV      34            /*  Get converter value. */

#define CMD_FC      35            /*  Filter configure.     */

#define CMD_CL      36            /*  Calibration routine   */
                                  /*  for A/D board.        */

#define CMD_BR      37            /*  Board write command. */
#define CMD_BW      38            /*  Board read command.   */

#define CMD_HELP    39            /*  Display key-help.     */

/*========================================================*/
/*  Type-int parameter table.                             */
/*--------------------------------------------------------*/
#define REG               1     /* Register most recently  */
                                /* read from or written    */
                                /* to.                     */
```

```
#define VALUE           2     /* Value most recently     */
                              /* used by a register-read */
                              /* or write.               */

#define IFACE           3     /* State of PCPI three-    */
                              /* state buffers.          */

#define BDINIT          4     /* State of -BDINIT line.  */

#define MRESET          5     /* State of -MRESET line.  */

#define SCREEN          6     /* State of screen-logger. */

#define LOG             7     /* State of session-       */
                              /* logger.                 */

#define LOG_NO          8     /* Number of current log   */
                              /* entry.                  */

#define LAST_LOG        9     /* Number of entry last    */
                              /* made to session-logger. */

#define QUIT            10    /* State of request to     */
                              /* terminate the present   */
                              /* session.                */

#define DATA_LEN        11    /* Length of valid data in */
                              /* data buffer.            */

#define LEN_INFILE      12    /* Length of valid data in */
                              /* input-data file.        */

#define LEN_OUTFILE     13    /* Length of valid data in */
                              /* last-written output-    */
                              /* data file.              */

#define FA_LC           14    /* Loop constant for A/D   */
                              /* conversion.             */

#define FD_LC           15    /* Loop constant for D/A   */
                              /* conversion.             */
```

```
/*===========================================================*/
/*  PCPI information and command structures.                 */
/*-----------------------------------------------------------*/
#define     L_LFN       30    /* Max. length of filename */
                              /* for session-log.        */
#define     L_LDATE     25    /* Length of date-string   */
                              /* for session-log.        */

#define     L_INFN      30    /* Maximum length of file- */
                              /* name for input-         */
                              /* data file.              */
#define     L_OUTFN     30    /* Maximum length of file- */
                             /* name for output-         */
                              /* data file.              */

typedef struct  pcpi_table
    {
    double       flt[21];     /* Type-double parameter   */
                              /* table.                  */
    int          fix[21];     /* Type-int parameter      */
                              /* table.                  */
    int          inreg[16];   /* Values contained in bus */
                              /* driver input-registers  */
                              /* (regs. written to by    */
                              /* computer/interface).    */
    int          outreg[8];   /* Values contained in bus */
                              /* driver output-registers */
                              /* (regs. read from by the */
                              /* computer/interface).    */
    double       con_d[11];   /* Type-double conversion  */
                              /* constants.              */
    int          con_i[11];   /* Type-int conversion     */
                              /* constants.              */
    unsigned     base_port;   /* Base port for register  */
                              /* address space.          */
    FILE         *logger;     /* Handle for session-     */
                              /* logger file.            */
    char         log_fn[L_LFN + 1];
                              /* Filename for session-   */
                              /* logger.                 */
    char         log_date[L_LDATE + 1];
                              /* Date for session-log.   */
    FILE         *infile;     /* Handle for input-data   */
                              /* file.                   */
```

```
    FILE          *outfile;       /* Handle for output-data   */
                                  /* file.                    */
    char          in_file[L_INFN + 1];
                                  /* Filename for input-data  */
                                  /* file.                    */
    char          out_file[L_OUTFN + 1];
                                  /* Filename for output-     */
                                  /* data file.               */
    } PCPI_TABLE;


typedef struct pcpi_command
    {
    char     cmd_str[5];          /* Keyboard-entry string    */
                                  /* for the command.         */
    char     log_format[80];      /* Format-string to be      */
                                  /* used by the session      */
                                  /* logger.                  */
    } PCPI_COMMAND;


/*=====================================================*/
/*  Definitions and declarations of static variables.  */
/*-----------------------------------------------------*/
#ifndef PI_INIT
    extern   PCPI_TABLE       pi;
    extern   PCPI_COMMAND     commands[MAX_CMDS];
    extern   char             pi_buf1[BUF_LEN];
    extern   char             pi_buf2[BUF_LEN];
    extern   char             pi_logbuf[BUF_LEN];
    extern   char             pi_tempbuf[BUF_LEN];
    extern   int              pi_data[MAX_DATA];

    extern   int              plot_len;       /*-----------*/
    extern   int              initialized[MAX_BDS];
    extern   int              trace;          /* ddn       */
    extern   int              stepper;        /* additions */
    extern   unsigned         num_pre_trig;   /*           */
    extern   unsigned         num_post_trig;  /*           */

    extern   int              test_das;       /* if YES,   */
                                              /* no bus    */
                                              /* i/o.      */
                                              /*-----------*/
```

```
#else
    PCPI_TABLE        pi;
    PCPI_COMMAND      commands[MAX_CMDS];
    char              pi_buf1[BUF_LEN];
    char              pi_buf2[BUF_LEN];
    char              pi_logbuf[BUF_LEN];
    char              pi_tempbuf[BUF_LEN];
    int               pi_data[MAX_DATA];
    int               plot_len;
    int               initialized[MAX_BDS];
    int               trace;
    int               stepper;
    unsigned          num_pre_trig;
    unsigned          num_post_trig;
    int               test_das;
#endif


/*===========================================================*/
/*  Function declarations.                                   */
/*-----------------------------------------------------------*/
#ifdef  LINT_ARGS

int     inp      (unsigned);
int     outp     (unsigned, int);


int     int_lnln(int, int, int, int, int, double, double,
                 double, int[], int, double, double,
                 double, int, char);
int     int_plot(int, int, int, int, int[], char[],
                 char[], char[], char[], char[], int,
                 double);
int     int_sc   (int[], int, int, double, int, double,
                 double *, double *, double *);


int     pi_cllog (void);
int     pi_cmd   (char *);
int     pi_cvt   (char *);

int     pi_date  (void);
int     pi_disp  (int, int *);
int     pi_gets  (int, char *);
```

```
int     pi_iface (int);
int     pi_init  (void);
int     pi_initl (int, int);
int     pi_log   (char *);
int     pi_opin  (char *);

int     pi_oplog (char *);
int     pi_opout (char *);
char    *pi_prsta(char *, char *, int);
int     pi_rdfil (void);
int     pi_rdreg (int);

int     pi_wrfil (void);
int     pi_wrreg (int, int);

#else


int     inp      ();
int     outp     ();

int     int_lnln ();
int     int_plot ();
int     int_sc   ();

int     pi_cllog ();
int     pi_cmd   ();
int     pi_cvt   ();

int     pi_date  ();
int     pi_disp  ();
int     pi_gets  ();

int     pi_iface ();
int     pi_init  ();
int     pi_initl ();
int     pi_log   ();
int     pi_opin  ();

int     pi_oplog ();
int     pi_opout ();
char    *pi_prsta();
int     pi_rdfil ();
int     pi_rdreg ();
```

```
int     pi_wrfil ();
int     pi_wrreg ();

#endif
```

```
/**********************************************************
 *
 *   SOURCE:          pi_das.c
 *
 *
 *   FUNCTION:        main()
 *
 *
 *   DESCRIPTION:     Mainline for the PI_DAS program.
 *                    Adapted from "pi.c" by Stephen A.
 *                    Dyer.  Used with permission.
 *
 *
 *   DOCUMENTATION
 *   FILES:           None.
 *
 *
 *   ARGUMENTS:       None.
 *
 *
 *   RETURN:          int
 *                    Zero.
 *
 *
 *   FUNCTIONS
 *   CALLED:          pi_cllog (),
 *                    pi_cmd   (),
 *                    pi_date  (),
 *                    pi_disp  (),
 *                    pi_gets  (),
 *                    pi_init  (),
 *                    pi_oplog ()
 *
 *
 *   AUTHOR:          Copyright 1989
 *                    Durwin D. Nigus
 *
 *
 *   DATE CREATED:    25 July 1989       Version 1.00
 *
 *
 *   REVISIONS:       None.
 *
 *
**********************************************************/
```

```
#include <errno.h>
#include <io.h>

#include <stdio.h>
#include <string.h>

#include <stdlib.h>
#include <process.h>

#include "pi_ddn.h"

int     errno;


main()
{
    int     cmd_code, error, board;

    int     err_access, lcl_errno;

/*  ---------------------------------------------------------*/
/*  Clear screen and print initial header on screen.        */
/*  ---------------------------------------------------------*/
    system ("cls");

    pi_date();
    printf("\n\n"
     "KANSAS STATE UNIVERSITY       "
     "Department of Electrical and Computer Engineering\n"
     "PC Parallel Interface (PCPI) Control Program \n"
     "------------------------------------------- \n"
     "     Data Acquisition System Controller \n"
     "         \n \%s\n\n", pi.log_date);



/*  ---------------------------------------------------------*/
/*  Initialize the interface and pertinent variables.       */
/*  ---------------------------------------------------------*/
    error = pi_init();

    switch (error)
       {
```

```
    case ERR_INITBP:
         printf("\nError on initialization:  "
           "no PCPI_BASE found in the environment.\n");
         return (0);
         break;

    case ERR_INITTST:
        printf ("\n"
                "no TEST_DAS found in the "
                "environment. \n\n"
                "Use TEST_DAS=0 for no test, \n\n"
                "TEST_DAS=10 for test mode. \n\n");
        return (0);
        break;

    case ERR_INITXTL:
        printf ("\n"
                "no XTAL_FREQ found in the "
                "environment. \n\n"
                "Use    set XTAL_FREQ=n \n\n"
                "where n is the A/D board crystal "
                "Frequency in Hz. \n\n\n");
        return (0);
        break;
    )

  printf("PCPI base port is %lu = %lXh.\n\n",
        pi.base_port, pi.base_port);

  printf ("A/D board oscillator frequency is %li Hz."
         "\n\n", xtal_freq);


  if (test_das == YES)
      printf ("ATTENTION! \n"
      "           DAS routine is in TEST mode "
      "e.g. all bus operations are reported \n"
      "           to the logger file. \n"
      "           To turn off test mode, set TEST_DAS "
      "to zero \n\n"
      "      from DOS. . .\n\n"
      "      set TEST_DAS=0 \n\n");
```

```
/*  ----------------------------------------------------*/
/*  Set up log file.                                    */
/*  ----------------------------------------------------*/
    for (;;)
        {
        printf("\nEnter filename for session log (%s):  ",

            pi.log_fn);
        pi_gets(L_LFN, pi_buf1);

        error = pi_oplog(pi_buf1);

        if (error == 0)
/*          Everything is okay.  Exit loop.                 */
            break;
        else if (error == ERR_OPEN)
            printf("\nError:  cannot open "
                   "session-log file.");
        else if (error == ERR_OPENACC)
            printf("\nError:  this session-log "
                   "file already exists.");
        }

/*-----------------------------------------------------*/
/*  Give brief instructions for setting up the external */
/*  module.                                             */
/*-----------------------------------------------------*/
    system ("cls");

    printf("\n\n\n    The PCPI bus has been disabled.\n\n"
        "1.  Make sure that power to the DAS "
        "is turned OFF.\n"
        "2.  Connect the PCPI bus cable to the interface "
        "board.\n"
        "3.  Turn power ON to the DAS.\n"
        "4.  Use the e(nable) command to enable the PCPI "
        "bus \n\n"
        "                    --or-- \n\n"
        "    use the 'si' command. \n\n"
        "NOTE:  Key-specific help is available by \n"
        "       typing the command  'help'.\n\n");
```

```
/*-----------------------------------------------------------*/
/*  Main command loop.                                       */
/*-----------------------------------------------------------*/
    for (;;)
        {
        printf("\npi:  ");
        pi_gets(50, pi_buf1);
        cmd_code = pi_cmd(pi_buf1);

        pi_disp(cmd_code, &board);

        pi_log(pi_logbuf);
        printf("%s\n", pi_logbuf);


/*      Check QUIT.                                          */
        if (pi.fix[QUIT])
            {
            pi_cllog();
            return (0);
            }

        }


    return (0);
}
```

PCPI-DAS SOURCE CODE: A/D BOARD COMMAND IMPLEMENTATION

```
/*************************************************************
 *
 *   SOURCE:           bi_das.c
 *
 *
 *   FUNCTION:         bi_das (board)
 *
 *
 *   DESCRIPTION:      The purpose of this function is to
 *                     simulate the "board initialize"
 *                     command issued to the DAS.
 *
 *
 *   DOCUMENTATION
 *   FILES:            None.
 *
 *
 *   ARGUEMENTS:
 *      board          (input) int
 *                     4-bit board address
 *
 *
 *   RETURN:           (int)
 *                     NORMAL:  normal return
 *
 *
 *   FUNCTIONS
 *   CALLED:           ctrl_wr (),
 *                     timer_wr (),
 *                     bd_wr ()
 *
 *
 *   AUTHOR:           Copyright 1989
 *                     Durwin D. Nigus
 *
 *
 *   DATE CREATED:     29 May 1989          Version 1.00
 *
 *   REVISIONS:        None.
 *
 *
 *************************************************************/
```

PCPI-DAS SOURCE CODE: A/D BOARD COMMAND IMPLEMENTATION

```
#include <stdio.h>
#include <ctype.h>

#include "pi_ddn.h"

#define FREQ_INIT       1000  /*  Initial freq., in Hz     */
#define NUM_SAMP_INIT   1000  /*  Initial # of samples.    */

int   bi_das (board)

int   board;
{
    unsigned    freq_init;
    float       freq_act;


/*  ======================================================*/
/*  Initialize the control/status register               */
/*  and set control registers to zero.                   */
/*  ------------------------------------------------------*/
    printf ("\n\n"
            "A/D board initialization.\n\n");

    bd_wr (board, CNT_STAT_CON,
           CON_STAT_CONF,REGULAR);

    bd_wr (board, CTRL1, 0, REGULAR);
    bd_wr (board, CTRL2, 0, REGULAR);

    printf ("Control-status register is set.\n");

/*  ------------------------------------------------------*/
/*  Reset the board error condition flag by setting an   */
/*  active conversion mode and returning to STANDBY      */
/*  (this transition resets many conditions on the       */
/*  A/D board).                                          */
/*  ------------------------------------------------------*/
    printf ("\nA/D board reset. \n");
    if (test_das == YES)
        fprintf (pi.logger, "\n A/D board reset. \n");

    ctrl_wr (board, BD_MODE, CONV_TR);
    ctrl_wr (board, BD_MODE, STBY);
```

```
/*  ==============================================================*/
/*  Set counter chip, 82C54-2                                      */
/*  --------------------------------------------------------------*/
    printf ("A/D conversion-delay timer set. \n");
    if (test_das == YES)
        fprintf (pi.logger, "Setting A/D conversion-delay"
                 " timer. \n");

    timer_wr (board, TMR_ONESHT, (unsigned)DLY_1SHT);

/*  --------------------------------------------------------------*/
/*  Store an arbitrary value in the sample counter                 */
/*  and sample period counter.                                     */
/*  --------------------------------------------------------------*/
    if (test_das == YES)
        fprintf (pi.logger, "Setting sample counter / "
                 "sample period generator. \n");

    timer_wr (board, TMR_SAMP_CNT,
             (unsigned)NUM_SAMP_INIT);

/*  Initialize the sample period generator with a value */
/*  that cooresponds to 1000 Hz.                          */
    freq_init = (unsigned)(xtal_freq / FREQ_INIT / 2.0) ;

    timer_wr (board, TMR_SAMP_PD, freq_init);

    freq_act = xtal_freq / freq_init / 2.0;

    printf ("\nSample couter set to   : %u samples,\n",
            (unsigned)NUM_SAMP_INIT);
    printf ("Sample frequency set to: %5.2f Hz.\n\n",
            freq_act);

/*  Modify the initialized-flag for this board address. */

    initialized[board] = YES;

    return (NORMAL);
}
```

PCPI-DAS SOURCE CODE: A/D BOARD COMMAND IMPLEMENTATION

```
/***********************************************************
 *
 *  SOURCE:          bp_das.c
 *
 *
 *  FUNCTION:        bp_das ()
 *
 *
 *  DESCRIPTION:     The purpose of this function is to
 *                   simulate the "determine boards present"
 *                   command issued to the DAS.
 *
 *
 *  DOCUMENTATION
 *  FILES:           None.
 *
 *
 *  ARGUEMENTS:      None.
 *
 *
 *  RETURN:          (int)
 *                   The number of boards found.
 *
 *
 *  FUNCTIONS
 *  CALLED:          bd_rd ()
 *
 *
 *  AUTHOR:          Copyright 1989
 *                   Durwin D. Nigus
 *
 *
 *  DATE CREATED:    29 May 1989          Version 1.00
 *
 *  REVISIONS:       None.
 *
 *
 ***********************************************************/
```

```
#include <stdio.h>
#include <ctype.h>

#include "pi_ddn.h"

int   bp_das ()

{
    int     bd_num,       /*   general-purpose counter     */
            num_pres,     /*   board-present counter       */
            test;

/*  ======================================================*/
/*  The procedure for determining the boards present      */
/*  is as follows:                                        */
/*                                                        */
/*        1)  set board address                           */
/*        2)  read EPROM board register (REG 0)           */
/*        3)  if bit 15 of REG 0 is LOW ("0"), a          */
/*            a board is present at this address.         */
/*                                                        */
/*  Repeat this for all possible board addresses.         */
/*                                                        */
/*  ------------------------------------------------------*/
    printf ("\n\n");
    num_pres = 0;

    for (bd_num = 0 ; bd_num < MAX_BDS ; bd_num++)
        {
        test = (int)bd_rd (bd_num, BD_PRESENT);

        if ((test & 0x80) == 0)
            {
            printf ("Board %d is present; ", bd_num);

            if (initialized[bd_num] == YES)
                printf ("  board is initialized. \n");
            else
                printf ("  board is NOT initialized.\n");

            num_pres++;
            }
        }
```

```
    if (num_pres != 0)
        printf ("\n\n"
                "There were %i board(s) found on the "
                "system bus. \n\n", num_pres);
    else
        printf ("No boards are present on the system "
                "bus.\n\n");

    return (num_pres);
}
```

PCPI-DAS SOURCE CODE: A/D BOARD COMMAND IMPLEMENTATION

```
/**************************************************************
 *
 *  SOURCE:         cc_das.c
 *
 *
 *  FUNCTION:       cc_das ()
 *
 *
 *  DESCRIPTION:    The purpose of this function is to
 *                  simulate the "system clock select"
 *                  command issued to the DAS.
 *
 *
 *  DOCUMENTATION
 *  FILES:          None.
 *
 *
 *  ARGUEMENTS:     None.
 *
 *
 *  RETURN:         (int)
 *                  NORMAL:  normal return
 *
 *
 *  FUNCTIONS
 *  CALLED:         bus_rd (),
 *                  sing_bit (),
 *                  bus_wr (),
 *                  pi_gets ()
 *
 *
 *  AUTHOR:         Copyright 1989
 *                  Durwin D. Nigus
 *
 *
 *  DATE CREATED:   29 May 1989         Version 1.00
 *
 *  REVISIONS:      None.
 *
 *
 **************************************************************/
```

PCPI-DAS SOURCE CODE: A/D BOARD COMMAND IMPLEMENTATION

```
#include <stdio.h>
#include <ctype.h>
#include <stdlib.h>

#include "pi_ddn.h"

int   cc_das ()
{
    char    prompt[PRMPT_MAX];

    int     menu,
            new_val;

/*========================================================*/
/*  SYSTEM TRIGGER SOURCE SELECTION                       */
/*--------------------------------------------------------*/
    printf ("SYSTEM CLOCK INPUT ENABLE . . . \n\n"
            "E) enable system clock input \n"
            "D) disable system clock input \n"
            "\n\n\n");

    pi_gets (PRMPT_MAX, prompt);
    menu = tolower (prompt[0]);

    new_val = bus_rd (BUS_CTRL);

/*--------------------------------------------------------*/
/*  Set the clock control bit accordingly (no-connect     */
/*  is the default)                                       */
/*--------------------------------------------------------*/
    if (menu == 'e')
        new_val = sing_bit (new_val, SYS_X_CLK_ID,
                            ACTIVE);
    else
        new_val = sing_bit (new_val, SYS_X_CLK_ID,
                            INACTIVE);

/*--------------------------------------------------------*/
/*  send the bus control value to the bus control         */
/*  register                                              */
/*--------------------------------------------------------*/
    bus_wr (BUS_CTRL, new_val);

    return (NORMAL);
}
```

PCPI-DAS SOURCE CODE: A/D BOARD COMMAND IMPLEMENTATION

```
/*************************************************************
 *
 *  SOURCE:          ct_das.c
 *
 *
 *  FUNCTION:        ct_das ()
 *
 *
 *  DESCRIPTION:     The purpose of this function is to
 *                   simulate the "system trigger select"
 *                   command issued to the DAS.
 *
 *
 *  DOCUMENTATION
 *  FILES:           None.
 *
 *
 *  ARGUEMENTS:      None.
 *
 *
 *  RETURN:          (int)
 *                   NORMAL:  normal return
 *
 *
 *  FUNCTIONS
 *  CALLED:          bus_rd (),
 *                   sing_bit (),
 *                   bus_wr (),
 *                   pi_gets ()
 *
 *
 *  AUTHOR:          Copyright 1989
 *                   Durwin D. Nigus
 *
 *
 *  DATE CREATED:    24 July 1989     Version 1.00
 *
 *  REVISIONS:       None.
 *
 *
*************************************************************/
```

PCPI-DAS SOURCE CODE: A/D BOARD COMMAND IMPLEMENTATION

```
#include <stdio.h>
#include <ctype.h>

#include "pi_ddn.h"

int   ct_das ()
{
    char     prompt[PRMPT_MAX];

    int      new_val,
             toggle_bit = 1 << (SYS_TR_LO_ID & 0x07);


/*  ====================================================*/
/*  System trigger source selection.                    */
/*  ----------------------------------------------------*/
    printf ("SYSTEM TRIGGER CONTROL  . . . \n\n"
            "A)  activate trigger (system ctrl'r) \n"
            "R)  pre-set for RISING-edge  trigger \n"
            "F)  pre-set for FALLING-edge trigger \n"
            "E)  connect external trigger \n"
            "D)  disconnect external trigger \n"
            "C)  bus-trigger controlled by board(s) \n"
            "\n\n"
            "selection??? ");

    pi_gets (PRMPT_MAX, prompt);

/*  ----------------------------------------------------*/
/*  Before executing any routines, retrieve the value   */
/*  of the bus control bus driver register.             */
/*  ----------------------------------------------------*/
    new_val = bus_rd (BUS_CTRL);


    switch (tolower (prompt[0]))
        {
        case 'c':
/*          --------------------------------------------*/
/*          Disable system control of system trigger    */
/*          i.e. switch to internal with INACTIVE       */
/*          logic level.                                */
/*          --------------------------------------------*/
            new_val = sing_bit (new_val, SYS_X_TRIG_ID,
                                ACTIVE);
```

```
          new_val = sing_bit (new_val,
                              SYS_TR_LO_ID, INACTIVE);

          bus_wr (BUS_CTRL, new_val);

          break;


       case 'r':
/*        ------------------------------------------------*/
/*        Pre-set for rising edge trigger sensitivity */
/*        i.e.  set to internal trigger source and    */
/*        set the trigger level to LOW (ACTIVE).      */
/*        ------------------------------------------------*/
          new_val = sing_bit (new_val, SYS_X_TRIG_ID,
                              INACTIVE);
          new_val = sing_bit (new_val,
                              SYS_TR_LO_ID, ACTIVE);

          bus_wr (BUS_CTRL, new_val);

          break;


       case 'f':
/*        ------------------------------------------------*/
/*        Pre-set for falling edge trigger            */
/*        sensitivity i.e. set to internal trigger    */
/*        source and set the trigger level to HIGH    */
/*        (INACTIVE).                                 */
/*        ------------------------------------------------*/
          new_val = sing_bit (new_val, SYS_X_TRIG_ID,
                              INACTIVE);
          new_val = sing_bit (new_val,
                              SYS_TR_LO_ID, INACTIVE);

          bus_wr (BUS_CTRL, new_val);

          break;
```

```
        case 'a':
/*          -----------------------------------------------*/
/*          Activate system trigger by toggling the       */
/*          system trigger control value.                 */
/*          -----------------------------------------------*/

            bus_wr (BUS_CTRL, new_val ^ toggle_bit);

            bus_wr (BUS_CTRL, new_val);

            break;


        case 'e':
/*          -----------------------------------------------*/
/*          Select the external trigger source.           */
/*          -----------------------------------------------*/
            new_val = sing_bit (new_val, SYS_X_TRIG_ID,
                                ACTIVE);

            bus_wr (BUS_CTRL, new_val);
            break;


        case 'd':
/*          -----------------------------------------------*/
/*          Select the internal trigger source (system    */
/*          controller).                                  */
/*          -----------------------------------------------*/
            new_val = sing_bit (new_val, SYS_X_TRIG_ID,
                                INACTIVE);

            bus_wr (BUS_CTRL, new_val);

            break;

       default:
            printf ("\n"
                    "No action taken. \n");
            break;
        }

    return (NORMAL);
}
```

```
/**********************************************************
 *
 *   SOURCE:          help_das.c
 *
 *
 *   FUNCTION:        help_das ()
 *
 *
 *   DESCRIPTION:     The purpose of this function is to
 *                    present keyh-specific help to the
 *                    PCPI/DAS user.
 *
 *
 *   DOCUMENTATION
 *   FILES:           None.
 *
 *
 *   ARGUEMENTS:      None.
 *
 *
 *   RETURN:          (int)
 *                    NORMAL:  Always.
 *
 *
 *   FUNCTIONS
 *   CALLED:          None.
 *
 *
 *   AUTHOR:          Copyright 1989
 *                    Durwin D. Nigus
 *
 *
 *   DATE CREATED:    11 August 1989     Version 1.00
 *
 *   REVISIONS:       None.
 *
 *
**********************************************************/
```

```
#include <stdio.h>
#include <stdlib.h>
#include <process.h>

#include "pi_ddn.h"

int   help_das ()
{

/*  --------------------------------------------------------*/
/*  All this routine does is execute the DOS type         */
/*  routine, where the typed file is "PCPI_DAS.HLP".      */
/*  --------------------------------------------------------*/
    system ("cls");
    if (system ("type pcpi_das.hlp | more") != 0)
        printf ("Help file, PCPI_DAS.HLP not in "
                "directory.\n\n");

    return (NORMAL);
}
```

```
/************************************************************
 *
 *  SOURCE:          si_das.c
 *
 *
 *  FUNCTION:        si_das ()
 *
 *
 *  DESCRIPTION:     the purpose of this function is to
 *                   configure the system at start up
 *                   (from the pi_das program).
 *
 *                   NOTE:  This routine verifies values
 *                   written to the bus drivers are
 *                   received.  If not, an abnormal program
 *                   termination results.
 *
 *
 *  DOCUMENTATION
 *  FILES:           None.
 *
 *
 *  ARGUEMENTS:      None.
 *
 *
 *  RETURN:          (int)
 *                   NORMAL:  no errors
 *
 *  FUNCTIONS
 *  CALLED:          cct_err (),
 *                   data_dir (),
 *                   bus_wr (),
 *                   pi_cllog ()
 *
 *
 *  AUTHOR:          Copyright 1989
 *                   Durwin D. Nigus
 *
 *
 *  DATE CREATED:    29 May 1989         Version 1.00
 *
 *  REVISIONS:       None.
 *
 *
************************************************************/
```

PCPI-DAS SOURCE CODE: A/D BOARD COMMAND IMPLEMENTATION

```
#include <stdio.h>
#include <ctype.h>
#include <process.h>
#include <stdlib.h>

#include "pi_ddn.h"

int   si_das ()
{
    int   ctrl;

    void  cct_err (void);


/*  Verify circuit is responding to PC.                              */
    bus_wr (PPI2_CTRL, BUS_ADDR);

    if (bus_rd (PPI2_CTRL) != BUS_ADDR)
        cct_err ();

    printf ("BUS initialization \n\n"
            "Data is in READ  mode, \n"
            "System trigger = connector \n"
            "System clock   = connector \n"
            "rd/wr inactive \n");

    ctrl = 0;
    ctrl = sing_bit (ctrl, BUS_RD_ID, INACTIVE);
    ctrl = sing_bit (ctrl, BUS_WR_ID, INACTIVE);
    ctrl = sing_bit (ctrl, SYS_X_TRIG_ID, ACTIVE);
    ctrl = sing_bit (ctrl, SYS_X_CLK_ID, ACTIVE);

    bus_wr (PPI2_CTRL, BUS_ADDR);
    data_dir (BUS_READ, 0, 0);
    bus_wr (BUS_CTRL, ctrl);

    return (NORMAL);
}
```

```
/***********************************************************
 *
 *  SOURCE:          bd_rd.c
 *
 *
 *  FUNCTION:        bd_rd (board, reg)
 *
 *
 *  DESCRIPTION:     The purpose of this function is to
 *                   read an 8 or 16-bit value from a
 *                   board located on the
 *                   "Acquisition System."
 *
 *
 *  DOCUMENTATION
 *  FILES:           None.
 *
 *
 *  ARGUEMENTS:
 *
 *     board         (input) int
 *                   4-bit value that selects the board
 *                   being addressed
 *
 *     reg           (input) int
 *                   4-bit value that selects the register
 *                   being addressed.  This also includes
 *                   information about the byte at which
 *                   the register resides.
 *
 *
 *  RETURN:          (unsigned)
 *                   the 8- or 16- bit number read from the
 *                   specified register
 *
 *  FUNCTIONS
 *  CALLED:          data_dir (),
 *                   bus_rd (),
 *                   rd_strob ()
 *
 *
 *  AUTHOR:          Copyright 1989
 *                   Durwin D. Nigus
 *
 *
```

```
 *   DATE CREATED:   7 August 1989        Version 1.00
 *
 *   REVISIONS:      None.
 *
 *
*********************************************************/
#include <stdio.h>
#include <ctype.h>

#include "pi_ddn.h"

unsigned   bd_rd (board, reg)

int    board, reg;

(
    unsigned     value;

/*  ----------------------------------------------------*/
/*  Set data direction and board/reg bus port.          */
/*  ----------------------------------------------------*/
    data_dir (BUS_READ, board, reg);

/*  ====================================================*/
/*  Determine if the register read is for LSB, MSB,     */
/*  or full 16-bit.                                     */
/*                                                      */
/*  When reading register value: activate the read      */
/*  line, read the data, then deactivate the read line. */
/*  ----------------------------------------------------*/

    switch (reg & DATA_MASK)
        {
        case DATA_LOW:
            rd_strob (ACTIVE, FAST);
            value = bus_rd (DAT_LSB);
            rd_strob (INACTIVE, FAST);
            break;

        case DATA_HIGH:
            rd_strob (ACTIVE, FAST);
            value = bus_rd (DAT_MSB);
            rd_strob (INACTIVE, FAST);
            break;
```

```
        case DATA_BOTH:
            rd_strob (ACTIVE, FAST);
            value = bus_rd (DAT_LSB) +
                    256 * bus_rd (DAT_MSB);
            rd_strob (INACTIVE, FAST);
            break;

        default:
            value = 0xffff; /* error in calling routine */
            break;
        }

    return (value);
}
```

```
/*************************************************************
 *
 *   SOURCE:          bd_wr.c
 *
 *
 *   FUNCTION:        bd_wr (board, reg, value, mode)
 *
 *
 *   DESCRIPTION:     The purpose of this function is to
 *                    write an 8- or 16- bit value to a
 *                    specified register on a
 *                    specified board.
 *
 *
 *   DOCUMENTATION
 *   FILES:           None.
 *
 *
 *   ARGUEMENTS:
 *
 *       board        (input) int
 *                    4-bit board address
 *
 *       reg          (input) int
 *                    4-bit register address
 *
 *       value        (input) unsigned
 *                    the 8- or 16- bit value to be written
 *                    to the appropriate register
 *
 *       mode         (input) int
 *                    selects between FAST and REGULAR.
 *
 *                    FAST:  data is written immediately;
 *                    REGULAR:  board and register are
 *                              written prior to write.
 *
 *
 *   RETURN:          (int)
 *                    NORMAL
 *
```

```
 *   FUNCTIONS
 *   CALLED:          byte_brk (),
 *                    data_dir (),
 *                    bus_wr (),
 *                    wr_strob ()
 *
 *
 *   AUTHOR:          Copyright 1989
 *                    Durwin D. Nigus
 *
 *
 *   DATE CREATED:    29 May 1989          Version 1.00
 *
 *
 *   REVISIONS:       None.
 *
 *
***********************************************************/
#include <stdio.h>
#include <ctype.h>

#include "pi_ddn.h"

int    bd_wr (board, reg, value, mode)

int    board,
       reg,
       mode;

unsigned    value;

{
    int     lsb, msb;

/*  ---------------------------------------------------*/
/*  Set the board and register address on the bus, and  */
/*  data direction if mode != FAST.                     */
/*  ---------------------------------------------------*/
    if (mode != FAST)
        data_dir (BUS_WRITE, board, reg);
```

```
/*   ---------------------------------------------------*/
/*   Write the data to appropriate data bus register.    */
/*   ---------------------------------------------------*/

/*   ---------------------------------------------------*/
/*   Determine the appropriate byte to which the data    */
/*   should be written. This information is in the most  */
/*   significant nibble of "reg" (masked by DATA_MASK).  */
/*   ---------------------------------------------------*/
     switch (reg & DATA_MASK)
         {
         case DATA_LOW:
             bus_wr (DAT_LSB, value);
             wr_strob ();
             break;

         case DATA_HIGH:
             bus_wr (DAT_MSB, value);
             wr_strob ();
             break;

         case DATA_BOTH:
/*           ---------------------------------------------*/
/*           Break the 16-bit value into its LSB and MSB */
/*           components. then write each.                */
/*           ---------------------------------------------*/
             byte_brk (value, &msb, &lsb);
             bus_wr (DAT_MSB, msb);
             bus_wr (DAT_LSB, lsb);
             wr_strob ();
             break;
         }

     return (NORMAL);
}
```

PCPI-DAS SOURCE CODE: BUS-DRIVER ROUTINES

```
/*************************************************************
 *
 *   SOURCE:          bus_rd.c
 *
 *
 *   FUNCTION:        bus_rd (reg)
 *
 *
 *   DESCRIPTION:     The purpose of this function is to
 *                    intercept read commands issued to
 *                    the PCPI and give the option to
 *                    echo PCPI READ operations on the
 *                    screen.
 *
 *
 *   DOCUMENTATION
 *   FILES:           None.
 *
 *
 *   ARGUEMENTS:
 *
 *      reg           (input) int
 *                    3-bit bus register to read
 *
 *
 *   RETURN:          (int)
 *                    the value read from the bus
 *
 *
 *   FUNCTIONS
 *   CALLED:          pi_rdreg ()
 *
 *
 *   AUTHOR:          Copyright 1989
 *                    Durwin D. Nigus
 *
 *
 *   DATE CREATED:    29 May 1989          Version 1.00
 *
 *   REVISIONS:       None.
 *
 *
 *************************************************************/
```

```
#include <stdio.h>
#include <ctype.h>
#include <string.h>
#include <stdlib.h>

#include "pi_ddn.h"

int   bus_rd (reg)

int   reg;
{
    int   resp;

    char  prompt[25], bin_buf[10];

/*  ----------------------------------------------------*/
/*  Read the register.                                  */
/*  ----------------------------------------------------*/
    resp = pi_rdreg (reg);


/*  ----------------------------------------------------*/
/*  Echo operation, if desired.                         */
/*  If test_das is true send record of operation to     */
/*  disk.                                               */
/*  If program is in the "trace" mode, echo the         */
/*  operation to screen.                                */
/*  ----------------------------------------------------*/
    ultoa ((long)resp, bin_buf, 2);

    if (test_das == YES)
        fprintf (pi.logger, "read  reg %2i value = "
                 "%4x h   %8s b\n", reg, resp, bin_buf);

    if (trace == YES)
        {

        printf ("read  reg %2i value = %4x h   %8s b\n",
                 reg, resp, bin_buf);
```

```
        if (stepper == YES)
            {
/*          ------------------------------------------*/
/*          During "step" mode, pause after each read.  */
/*          ------------------------------------------*/
            printf ("     <enter> to continue "
                    "[ end step <enter> to cease step]"
                    " . . . \n");
            pi_gets (25, prompt);
            if (strcmp (prompt, "end step") == 0)
                stepper = NO;
            }
        }

    return (resp);
}
```

PCPI-DAS SOURCE CODE: BUS-DRIVER ROUTINES

```
/*************************************************************
 *
 *  SOURCE:          bus_wr.c
 *
 *
 *  FUNCTION:        bus_wr (bus_reg, value)
 *
 *
 *  DESCRIPTION:     The purpose of this function is to
 *                   intercept PCPI writes and give the
 *                   option for screen echo of write
 *                   operation.
 *
 *
 *  DOCUMENTATION
 *  FILES:           None.
 *
 *
 *  ARGUEMENTS:
 *
 *     bus_reg       (input) int
 *                   3-bit bus register address
 *
 *     value         (input) int
 *                   The 8-bit value to be written
 *                   to the bus register
 *
 *
 *  RETURN:          (int)
 *                   NORMAL:  normal return
 *
 *
 *  FUNCTIONS
 *  CALLED:          pi_wrreg ()
 *
 *
 *  AUTHOR:          Copyright 1989
 *                   Durwin D. Nigus
 *
 *
 *  DATE CREATED:    29 May 1989          Version 1.00
 *
 *  REVISIONS:       None.
 *
 *
```

```
*************************************************************/
#include <stdio.h>
#include <ctype.h>
#include <string.h>
#include <stdlib.h>

#include "pi_ddn.h"

int   bus_wr (bus_reg, value)

int   bus_reg, value;
{

    char   prompt[25], bin_buf[10];

/*  ---------------------------------------------------*/
/*  If TEST_DAS is NOT defined in the environment,     */
/*  execute the register-write operation.              */
/*  ---------------------------------------------------*/
    pi_wrreg (bus_reg, value);


/*  ---------------------------------------------------*/
/*  If test_das is true, send a record of the operation */
/*  to disk.                                           */
/*  If program is in the "trace" mode, echo the        */
/*  operation to screen.                               */
/*  ---------------------------------------------------*/
    if (trace == YES || test_das == YES)
        {
        ultoa ((long)value, bin_buf, 2);

        if (test_das == YES)
            {
            fprintf (pi.logger, "write reg %2i value "
                     "= %4x h   %8s b ", bus_reg, value,
                     bin_buf);

            if (bus_reg == REG_ADR)
                fprintf (pi.logger, "  WRITE to board "
                         "reg %2i ", value/16);

            fprintf (pi.logger, " \n");
            }
```

```
        if (trace == YES)
            {
            printf ("write reg %2i value "
                    "= %4x h   %8s b\n", bus_reg, value,
                    bin_buf);

            if (bus_reg == REG_ADR)
                printf ("  set to board "
                         "reg %2i ", value/16);

            printf ("\n");
            }

            if (stepper == YES)
                {
/*              ----------------------------------------*/
/*              If program is in "step" mode, pause.    */
/*              ----------------------------------------*/
                printf ("    <enter> to continue,"
                    "[ end step <enter> to cease step]"
                    " . . . \n");
                pi_gets (25, prompt);
                if (strcmp (prompt, "end step") == 0)
                    stepper = NO;
                }
        }

    return (NORMAL);
}
```

```
/***********************************************************
 *
 *   SOURCE:          data_dir.c
 *
 *
 *   FUNCTION:        data_dir (state, board, reg)
 *
 *
 *   DESCRIPTION:     The purpose of this function is to set
 *                    the data direction on the system bus.
 *
 *
 *   DOCUMENTATION
 *   FILES:           None.
 *
 *
 *   ARGUEMENTS:
 *
 *      state         (input) int
 *                    Select the direction of the data bus;
 *                    either
 *                            BUS_READ
 *                            BUS_WRITE
 *
 *      board         (input) int
 *                    4-bit board addresss to which will be
 *                    written to or read from
 *
 *      reg           (input) int
 *                    4-bit register address which commun-
 *                    ication is intended
 *
 *   RETURN:          (int)
 *                        NORMAL  if no errors detected;
 *                        ERR     if error detected.
 *
 *
 *   FUNCTIONS
 *   CALLED:          pause (),
 *                    bus_rd (),
 *                    bus_wr (),
 *                    cct_err (),
 *                    pi_cllog (),
 *                    pi_iface ()
 *
```

```
 *
 *  AUTHOR:          Copyright 1989
 *                   Durwin D. Nigus
 *
 *
 *  DATE CREATED:    28 May 1989           Version 1.00
 *
 *  REVISIONS:       None.
 *
 *
*********************************************************/
#include <stdio.h>
#include <ctype.h>

#include "pi_ddn.h"

int     data_dir (state, board, reg)

int     state, board, reg;
{
    char    prompt[PRMPT_MAX];

    int     combined, new, prev;

    void    cct_err (void);

/*  ---------------------------------------------------*/
/*  Verify bus read / write strobe lines are inactive  */
/*  before changing the direction.                     */
/*  ---------------------------------------------------*/
    prev = bus_rd (BUS_CTRL);
    new  = sing_bit (prev, BUS_RD_ID, INACTIVE);
    new  = sing_bit (new, BUS_WR_ID, INACTIVE);
    bus_wr (BUS_CTRL, new);


/*  -------------------------------------------------*/
/*  State of data bus is BUS_READ or BUS_WRITE.      */
/*  -------------------------------------------------*/
    bus_wr (DAT_CTL_REG, state);
```

```
/*  ---------------------------------------------------------*/
/*  Set the board and register address on the bus.          */
/*  Mask the data byte information from 'reg' with          */
/*  0x0f hex.                                                */
/*  ---------------------------------------------------------*/
    reg = reg & -DATA_MASK;
    combined = board * BD_MULT + reg * REG_MULT;
    bus_wr (REG_ADR, combined);

/*  ---------------------------------------------------------*/
/*  Verify data direction register setting.                 */
/*  If an error exists, terminate program.                  */
/*  ---------------------------------------------------------*/
    if (bus_rd (DAT_CTL_REG) != state)
        cct_err ();

    return (NORMAL);
}
```

```
/************************************************************
 *
 *   SOURCE:          cct_err.c
 *
 *
 *   FUNCTION:        cct_err ()
 *
 *
 *   DESCRIPTION:     The purpose of this function is to
 *                    cease the operation of the program,
 *                    usually if the circuit is not
 *                    responding to I/O from the PC.
 *
 *
 *   DOCUMENTATION
 *   FILES:           None.
 *
 *
 *   ARGUEMENTS:      None.
 *
 *
 *   RETURN:          None -- Program is terminated.
 *
 *
 *   FUNCTIONS
 *   CALLED:          pi_cllog (),
 *                    pi_iface ()
 *
 *
 *   AUTHOR:          Copyright 1989
 *                    Durwin D. Nigus
 *
 *
 *   DATE CREATED:    23 August 1989          Version 1.00
 *
 *   REVISIONS:       None.
 *
 *
 ************************************************************/
```

```
#include <stdio.h>
#include <stdlib.h>

#include "pi_ddn.h"

void   cct_err ()
{
/*  ----------------------------------------------------*/
/*  Disable the interface, put a message in the logger  */
/*  and abort the program.                               */
/*  ----------------------------------------------------*/
    printf ("\n ERROR!!! \n"
            "\n Interface has been disabled.\n "
            " Program has been terminated.\n\n"
            " Error may be caused by no power to circuit."
            "\n\n\n");

    pi_iface (DISABLE);

    sprintf (pi_logbuf, "Error with interface circuit. "
             " Program terminated.");

    pi_cllog ();
    abort ();
}
```

```
/***********************************************************
 *
 *  SOURCE:          rd_strob.c
 *
 *
 *  FUNCTION:        rd_strob (state, mode)
 *
 *
 *  DESCRIPTION:     The purpose of this function is to
 *                   control the bus read strobe.
 *
 *
 *  DOCUMENTATION
 *  FILES:           None.
 *
 *
 *  ARGUEMENTS:
 *
 *     state         (input) int
 *                   sets the read strobe to either:
 *
 *                     ACTIVE   :  read strobe active
 *                     INACTIVE :  read strobe inactive
 *
 *     mode          (input) int
 *                   selects between a normal strobe (with
 *                   data direction checking) and a fast
 *                   strobe (during repeatitive reads)
 *
 *                     NORMAL   :  check data bus direction
 *                     FAST     :  modify read strobe
 *                                 immediately
 *
 *
 *  RETURN:          (int)
 *                   ERR    :  read requested when bus
 *                             is in WRITE mode
 *                   NORMAL:   read strobe is in specified
 *                             position
 *
 *  FUNCTIONS
 *  CALLED:          bus_rd (),
 *                   bus_wr (),
 *                   cct_err (),
 *                   sing_bit ()
```

```
 *
 *
 *  AUTHOR:          Copyright 1989
 *                   Durwin D. Nigus
 *
 *
 *  DATE CREATED:    28 May 1989          Version 1.00
 *
 *  REVISIONS:       None.
 *
 *
**********************************************************/
#include <stdio.h>
#include <ctype.h>

#include "pi_ddn.h"

int   rd_strob (state, mode)

int   state, mode;
{
    int     prev_val, new_val;

    char    prompt[5];

    void    cct_err (void);

/*   ----------------------------------------------------*/
/*   Verify data direction doesn't cause data bus        */
/*   contention (if active is requested).                */
/*   ----------------------------------------------------*/
    if ((state == ACTIVE) && (mode != FAST))
       {
       if (bus_rd (DAT_CTL_REG) != BUS_READ)
           {
           printf ("data bus contention ERROR \n"
                   "press ctrl-C to stop program ");
           scanf  ("%s", prompt);
           return (ERR);
           }
        }
```

```
/*  ------------------------------------------------------*/
/*  retrieve a copy of board control register prior to  */
/*  sending control value,                               */
/*  set write bit INACTIVE,                              */
/*  set read bit in accordance with   state              */
/*  ------------------------------------------------------*/
    prev_val = bus_rd (BUS_CTRL);
    prev_val = sing_bit (prev_val, BUS_WR_ID, INACTIVE);
    new_val  = sing_bit (prev_val, BUS_RD_ID, state);

    bus_wr (BUS_CTRL, new_val);    /* write state   */

/*  ------------------------------------------------------*/
/*  Verify bus control is desired value.                 */
/*  If error exists, terminate program.                  */
/*  ------------------------------------------------------*/
    if (bus_rd (BUS_CTRL) != new_val)
        cct_err ();

    return (NORMAL);
}
```

```
/**************************************************************
 *
 *   SOURCE:         wr_strob.c
 *
 *
 *   FUNCTION:       wr_strob ()
 *
 *
 *   DESCRIPTION:    The purpose of this function is to
 *                   make the bus write strobe active
 *                   for a brief period of time.  If, dur-
 *                   ing the restoration, the bus
 *                   control register does not return to
 *                   its previous value, a error is
 *                   generated.
 *
 *
 *   DOCUMENTATION
 *   FILES:          None.
 *
 *
 *   ARGUEMENTS:     None.
 *
 *
 *   RETURN:         (int)
 *                   NORMAL   :  normal return
 *                   ERR      :  bus control register error
 *
 *   FUNCTIONS
 *   CALLED:         bus_rd (),
 *                   bus_wr (),
 *                   cct_err ()
 *
 *
 *   AUTHOR:         Copyright 1989
 *                   Durwin D. Nigus
 *
 *
 *   DATE CREATED:   28 May 1989        Version 1.00
 *
 *   REVISIONS:      None.
 *
 *
 **************************************************************/
```

```
#include <stdio.h>
#include <ctype.h>

#include "pi_ddn.h"

int   wr_strob ()
{
    int      prev_val, wr_val;

    char     prompt[5];

    void     cct_err (void);

/*  -----------------------------------------------------*/
/*  Sending a WRITE strobe is simply a matter of making */
/*  the bus write line active, and one instruction      */
/*  later making it inactive.  It is also important to  */
/*  verify the contents of the bus control register     */
/*  before exiting this routine.                        */
/*  -----------------------------------------------------*/

/*  -----------------------------------------------------*/
/*  Retrieve a copy of the board control register       */
/*  verify read and write bit were inactive.            */
/*  -----------------------------------------------------*/
    prev_val = bus_rd (BUS_CTRL);
    prev_val = sing_bit (prev_val, BUS_RD_ID, INACTIVE);
    prev_val = sing_bit (prev_val, BUS_WR_ID, INACTIVE);
    wr_val   = sing_bit (prev_val, BUS_WR_ID, ACTIVE);

/*  -----------------------------------------------------*/
/*  Send the strobe.                                    */
/*  -----------------------------------------------------*/
    bus_wr (BUS_CTRL, wr_val);   /*  Make write active, */
    bus_wr (BUS_CTRL, prev_val); /*  restore bus        */
                                 /*  control.           */

/*  -----------------------------------------------------*/
/*  Verify bus is previous value (RD/WR inactive).      */
/*  -----------------------------------------------------*/
    if (bus_rd (BUS_CTRL) != prev_val)
        cct_err ();

    return (NORMAL);
}
```

```
/*************************************************************
 *
 *   SOURCE:          bc_das.c
 *
 *
 *   FUNCTION:        bc_das (board)
 *
 *
 *   DESCRIPTION:     The purpose of this function is to
 *                    simulate the "begin conversion" command
 *
 *
 *   DOCUMENTATION
 *   FILES:           None.
 *
 *
 *   ARGUEMENTS:
 *
 *      board         (input) int
 *                    4-bit board address
 *
 *
 *   RETURN:          (int)
 *                    NORMAL:  normal return.
 *                    ERR   :  illegal number of samples;
 *                             board all ready active
 *
 *
 *   FUNCTIONS
 *   CALLED:          ad_test (),
 *                    bd_rd (),
 *                    ctrl_wr (),
 *                    give_val (),
 *                    pi_cvt (),
 *                    pi_gets (),
 *                    timer_wr ()
 *
 *
 *
 *   AUTHOR:          Copyright 1989
 *                    Durwin D. Nigus
 *
 *
 *   DATE CREATED:    29 May 1989         Version 1.00
 *
```

```
 *  REVISIONS:     None.
 *
 *
**********************************************************/
#include <stdio.h>
#include <ctype.h>
#include <stdlib.h>

#include "pi_ddn.h"

int    bc_das (board)

int    board;
{
    char        prompt[PRMPT_MAX];

    int         c1, conv_mode, dummy, i, protect;

    unsigned    max, max_post, min, temp_samp;


/*  ====================================================*/
/*  Determine if board is all ready in conversion mode. */
/*  ----------------------------------------------------*/

/* ----- retrieve CONTROL #1  ---------------------------*/
    c1  = (int)bd_rd (board, CTRL1);

    if ( give_val (BD_MODE, c1) != STBY)
        {
        printf ("\n\n"
                "Board is all ready active. \n\n"
                "Use Stop Conversion (sc) command before "
                "using this command. \n\n");
        return (ERR);
        }

/*  ====================================================*/
/*  Determine conversion mode.                          */
/*  ----------------------------------------------------*/
    printf ("\nSelect:  \n\n"
            "I)  immediate conversion \n"
            "W)  wait for trigger \n"
            "\n\n\n"
            "your selection: ");
```

```
    pi_gets (PRMPT_MAX, prompt);


/*  =================================================*/
/*  Set the number of samples.                        */
/*  -------------------------------------------------*/
    switch (tolower (prompt[0]))
        {
        case 'w':
/*          ---------------------------------------*/
/*          Determine if pre-trigger sampling should  */
/*          occur.                                    */
/*          ---------------------------------------*/
            printf ("\n\n"
                    "number of pre-trigger samples [max"
                    " = 65535] (%u):  ", num_pre_trig);
            if (pi_gets (PRMPT_MAX, prompt) != 0)
                num_pre_trig = (unsigned)pi_cvt (prompt);

/*          Is pre-trigger acquisition desired?         */
            conv_mode = (num_pre_trig == 0) ?
                        (CONV_TR) : (CONV_TR_PRE);

            max_post = 65535 - num_pre_trig;
            printf ("\n\n"
                    "number of post-trigger samples "
                    "[max = %u] (%u): ",
                    max_post, num_post_trig);
            if (pi_gets (PRMPT_MAX, prompt) != 0)
                num_post_trig = (unsigned)pi_cvt (prompt);

            break;


        case 'i':
/*          ---------------------------------------*/
/*          Immediate data acquisition (no trigger).  */
/*          ---------------------------------------*/
            conv_mode = CONV_NOW;
            printf ("\n\n"
                  "number of samples [max = 65535] (%u):"
                  "  ", num_post_trig);
            if (pi_gets (PRMPT_MAX, prompt) != 0)
```

```
                num_post_trig = (unsigned)pi_cvt (prompt);

            if (num_post_trig != 0)
                num_pre_trig  = 0;
            break;


        default:
            printf ("\n"
                    "No conversions requested.\n\n");
            return (ERR);
        }

/*  =======================================================*/
/*  Varify the sum of pre- and post- trigger samples       */
/*  is not greater than memory limitations (64K), and      */
/*  not equal to zero.                                     */
/*  -------------------------------------------------------*/
    if ((num_pre_trig + num_post_trig) == 0)
        {
        printf ("No samples requested. \n\n");
        return (ERR);
        }

    min = (num_post_trig > num_pre_trig) ?
          (num_pre_trig) : (num_post_trig);
    max = (num_post_trig > num_pre_trig) ?
          (num_post_trig): (num_pre_trig);

    if ((65535 - max) < min)
        {
        printf ("Illegal total number of samples "
                "requested. \n\n"
                " --- routine aborted --- \n\n");
        return (ERR);
        }

/*  =======================================================*/
/*  Set-up A/D board control registers appropriately.      */
/*                                                         */
/*  The sample counter must be written to first, then      */
/*  the conversion mode should be set.                     */
/*                                                         */
```

```
/*  Note that one is subtraced from the number of post- */
/*  trigger samples.  The A/D board requires this       */
/*  counter to be one less than the number of samples.  */
/*                                                      */
/*  The sample counter operates in such a way that      */
/*  the minimum number of counts is 1 (set the counter  */
/*  to 1) and the maximum number of counts is 2^16      */
/*  (set the counter to 0).                             */
/*  ----------------------------------------------------*/

    temp_samp = (num_post_trig == 0)
                ? (1) : (num_post_trig);

    temp_samp = (num_post_trig == 1)
                ? (1) : (num_post_trig);

    temp_samp = (num_post_trig == 65535)
                ? (0) : (num_post_trig);


/*  Determine if the protection circuit is enabled.     */
    printf ("\n\n"
            "Overload protection circuit enabled? "
            "(Y/N) ");
    pi_gets (PRMPT_MAX, prompt);

    protect = (tolower (prompt[0]) == 'n') ?
              (PROTECT_OFF) : (PROTECT_ON);
    ctrl_wr (board, PROTECT, protect);


    if (protect == PROTECT_ON)
        printf ("\nInput protection enabled.\n\n");
    else
        printf ("\nCAUTION!  Input protection "
                "disabled.\n\n");

/*  Connect the signal.                                 */
    ctrl_wr (board, SIG_SEL, SIG_APPLIED);

/*  Delay;  wait for the signal connection relays       */
/*  to activate.                                        */
```

```
    for (i = 0 ; i < 5000 ; i++)
        {
        dummy = i;
        }

/*  Write number of post-trig samples to counter.          */
    timer_wr (board, TMR_SAMP_CNT, temp_samp);

/*  Write converison mode to control register.             */
    ctrl_wr (board, BD_MODE, conv_mode);

    return (NORMAL);
}
```

```
/*************************************************************
 *
 *  SOURCE:          cl_das.c
 *
 *
 *  FUNCTION:        cl_das (board)
 *
 *
 *  DESCRIPTION:     The purpose of this function is to
 *                   enable the user to calibrate the A/D
 *                   board.
 *
 *
 *  DOCUMENTATION
 *  FILES:           None.
 *
 *
 *  ARGUEMENTS:
 *
 *     board         (input) int
 *                   4-bit board address
 *
 *
 *  RETURN:          (int)
 *                   NORMAL:  normal return.
 *                   ERR   :  abnormal return.
 *
 *
 *  FUNCTIONS
 *  CALLED:          ctrl_wr (),
 *                   bd_rd (),
 *                   bd_wr (),
 *                   give_val (),
 *                   pi_gets ()
 *
 *
 *  AUTHOR:          Copyright 1989
 *                   Durwin D. Nigus
 *
 *
 *  DATE CREATED:    27 June 1989        Version 1.00
 *
 *  REVISIONS:       None.
 *
 *
```

```
************************************************************/
#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>

#include "pi_ddn.h"

int     cl_das (board)

int     board;
{
    char      prompt[PRMPT_MAX], bin_buf[25];

    int       c1, conv_mode, menu;

    unsigned  ad_val, count, num_conv;

/*=========================================================*/
/*  determine if board is all ready in conversion mode    */
/*---------------------------------------------------------*/

/*  retrieve CONTROL #1                                    */
    c1  = (int)bd_rd (board, CTRL1);
    conv_mode  = give_val (BD_MODE, c1);

    if (conv_mode != STBY)  /*       error if true         */
        {
        printf ("\n\n"
                "Board is all ready active. \n\n"
                "Use Stop Coversion command before "
                "using this command. \n\n");
        return (ERR);
        }

/*=========================================================*/
/*  determine calibration mode                             */
/*---------------------------------------------------------*/
    printf ("\nSelect:  \n\n"
            "T) x10 gain adjustment  \n"
            "A) A/D adjustment \n"
            " --- any other selection aborts routine"
            " --- \n\n"
            "your selection: ");

    pi_gets (PRMPT_MAX, prompt);
```

```
    menu = tolower (prompt[0]);

/*-------------------------------------------------------*/
/*  execute the apporpriate calibration routine          */
/*-------------------------------------------------------*/
    switch (menu)
        {
        case 't':
/*-------------------------------------------------------*/
/*          x10 gain adjustment routine                  */
/*          1)  turn on overload protection              */
/*          2)  set gain to ten                          */
/*          3)  connect the signal relays                */
/*              . . . wait for stop command . . .        */
/*          4)  decouple the signal relays               */
/*                     exit                              */
/*                                                       */
/*   NOTE:  These adjustments are made without the A/D! */
/*                                                       */
/*-------------------------------------------------------*/
            ctrl_wr (board, PROTECT, PROTECT_ON);
            ctrl_wr (board, GAIN_SEL, GAIN_10);
            ctrl_wr (board, SIG_SEL, SIG_APPLIED);

            printf ("\n"
                    "Gain x10 adjustment . . . press "
                    "<enter> to stop. ");
            pi_gets (PRMPT_MAX, prompt);
            break;

        case 'a':
/*-------------------------------------------------------*/
/*          The A/D adjustment routine.                  */
/*          1)  turn on protection                       */
/*          2)  select gain = 1                          */
/*          3)  enable the signal relays                 */
/*          4)  prompt for the number of samples for     */
/*              conversion sequence . . .                */
/*          5)  Initiate conversion                      */
/*          6)  Check the EOC status bit and retrieve    */
/*              data when ACTIVE                         */
```

```
/*          7)  display the number on the screen              */
/*              and repeat steps  5, 6, 7 until the           */
/*              number of samples requested in (4)            */
/*              have been acquired.                           */
/*------------------------------------------------------------*/
            ctrl_wr (board, PROTECT, PROTECT_ON);
            ctrl_wr (board, GAIN_SEL, GAIN_1);
            ctrl_wr (board, SIG_SEL, SIG_APPLIED);

            for (;;)
                {
                printf ("\n"
                    "A/D converter calibration routine."
                    "\n\n"
                    "How many samples to be acquired"
                    " during calibration?"
                    "\n\n"
                    "respond with 0 to stop: ");

                pi_gets (PRMPT_MAX, prompt);
                num_conv = (unsigned)pi_cvt (prompt);
                printf ("\n\n%5u samples selected.\n",
                        num_conv);

/*              Stop routine when finished.                   */
                if (num_conv == 0)
                    {
                    ctrl_wr (board, SIG_SEL, SIG_REMOVE);
                    return (NORMAL);
                    }


                bd_wr (board, AD_DIRECT,
                       0, REGULAR);
                for (count = num_conv ;
                     count > 0 ; count--)
                    {
                    bd_wr (board, AD_DIRECT,
                           0, FAST);


/*                  Wait for end-of-conversion.               */
                    c1  = EOC_ACT + 1;
```

```
                        while (give_val (EOC, c1) != EOC_ACT)
                            {
                            c1   = (int)bd_rd (board, STATUS);
                            }


/*                      Read the sample value.                   */

                        ad_val = bd_rd (board, AD_DIRECT);

/*                      Convert to binary.                       */
                        ultoa ((long)ad_val, bin_buf, 2);

/*                      Display.                                 */
                        printf ("\n"
                                "conv. value = %4x (hex)"
                                "  %16sb", ad_val, bin_buf);
                        }
                    }
                break;
            }

/*-------------------------------------------------------*/
/*  routines complete, disconnect signal                 */
/*-------------------------------------------------------*/
    ctrl_wr (board, SIG_SEL, SIG_REMOVE);

    return (NORMAL);
}
```

```
/***********************************************************
 *
 *  SOURCE:         cs_das.c
 *
 *
 *  FUNCTION:       cs_das (board)
 *
 *
 *  DESCRIPTION:    The purpose of this function is to
 *                  simulate the "clock select"
 *                  command issued to the DAS.
 *
 *
 *  DOCUMENTATION
 *  FILES:          None.
 *
 *
 *  ARGUEMENTS:
 *     board        (input) int
 *                  4-bit board address
 *
 *
 *  RETURN:         (int)
 *                  NORMAL:  normal return
 *
 *
 *  FUNCTIONS
 *  CALLED:         ctrl_wr (),
 *                  pi_gets ()
 *
 *
 *  AUTHOR:         Copyright 1989
 *                  Durwin D. Nigus
 *
 *
 *  DATE CREATED:   29 May 1989          Version 1.00
 *
 *  REVISIONS:      None.
 *
 *
***********************************************************/
```

```
#include <stdio.h>
#include <ctype.h>

#include "pi_ddn.h"

int     cs_das (board)

int     board;
{
      char prompt[PRMPT_MAX];

/*  ====================================================*/
/*  Clock source selection for A/D board.               */
/*  ----------------------------------------------------*/
     printf ("A/D board:  clock selection . . . \n\n"
             "I) internal \n"
             "B) bus  \n"
             "P) front panel  \n"
             "\n\n"
             "selection???  ");

    pi_gets (PRMPT_MAX, prompt);

    switch (tolower (prompt[0]))
        {
        case 'i':
/*          ----------------------------------------*/
/*          Internal clock selected.  Although this is  */
/*          not part of the 'ct' command for the DAS,   */
/*          this routine calls the 'sr' command to      */
/*          enable the user to select the frequency.    */
/*          ----------------------------------------*/
            sr_das (board);
            break;

        case 'b':
            ctrl_wr (board, CLOCK_SEL, CLK_BUS);
            break;

        case 'p':
            ctrl_wr (board, CLOCK_SEL, CLK_PANEL);
            break;
```

```
        default:
            printf ("\n\n"
                    "No action taken.");
            break;
        }

    return (NORMAL);
}
```

```
/**************************************************************
 *
 *  SOURCE:          dr_das.c
 *
 *
 *  FUNCTION:        dr_das (board)
 *
 *
 *  DESCRIPTION:     The purpose of this function is to
 *                   retrieve the bus driver values
 *
 *
 *  DOCUMENTATION
 *  FILES:           None.
 *
 *
 *  ARGUEMENTS:      None.
 *
 *
 *  RETURN:          (int)
 *                   NORMAL:  normal return
 *
 *
 *  FUNCTIONS
 *  CALLED:          bus_rd ()
 *
 *
 *  AUTHOR:          Copyright 1989
 *                   Durwin D. Nigus
 *
 *
 *  DATE CREATED:    29 May 1989          Version 1.00
 *
 *  REVISIONS:       None.
 *
 *
 **************************************************************/
```

```
#include <stdio.h>
#include <ctype.h>


#include "pi_ddn.h"

int   dr_das ()
{
    int   reggie, reg_val;

/*   ====================================================*/
/*   Display contents of bus driver registers.           */
/*   ----------------------------------------------------*/
     printf ("\n\n"
             "Bus driver register values . . . \n\n");

     reggie = DAT_LSB;
     reg_val = bus_rd (reggie);
     printf ("data LSB ,   reg %1d, value = %2X h "
             "(%3d dec) \n", reggie, reg_val, reg_val);

     reggie = DAT_MSB;
     reg_val = bus_rd (reggie);
     printf ("data MSB ,   reg %d, value = %2X h "
             "(%3d dec)\n", reggie, reg_val, reg_val);

     reggie = ADDR_LSB;
     reg_val = bus_rd (reggie);
     printf ("addr LSB ,   reg %d, value = %2X h "
             "(%3d dec)\n", reggie, reg_val, reg_val);

     reggie = ADDR_MSB;
     reg_val = bus_rd (reggie);
     printf ("addr MSB ,   reg %d, value = %2X h "
             "(%3d dec)\n", reggie, reg_val, reg_val);

     reggie = BUS_CTRL;
     reg_val = bus_rd (reggie);
     printf ("bus ctrl ,   reg %d, value = %2X h "
             "(%3d dec) \n", reggie, reg_val, reg_val);
```

```
    reggie = REG_ADR;
    reg_val = bus_rd (reggie);
    printf ("bd/reg addr, reg %d, value = "
            "%2X h (%3d dec) \n",
            reggie, reg_val, reg_val);

    reggie = PPI1_CTRL;
    reg_val = bus_rd (reggie);
    printf ("\nPPI1 ctrl (data and bd/reg)  "
            "[read = 92h, write = 80h] \n"
            "reg %d, value = %2X h (%3d dec) \n\n",
            reggie, reg_val, reg_val);

    reggie = PPI2_CTRL;
    reg_val = bus_rd (reggie);
    printf ("PPI2 ctrl (address and bus ctrl)  "
            "[should be 80h] \n"
            "reg %d, value = %2X h (%3d dec) \n\n",
            reggie, reg_val, reg_val);
    printf ("\n\n");

    return (NORMAL);
}
```

```
/*************************************************************
 *
 *  SOURCE:          fc_das.c
 *
 *
 *  FUNCTION:        fc_das (board)
 *
 *
 *  DESCRIPTION:     The purpose of this function is to
 *                   simulate the "filter configure"
 *                   command issued to the DAS.
 *
 *
 *
 *  DOCUMENTATION
 *  FILES:           None.
 *
 *
 *  ARGUEMENTS:
 *     board         (input) int
 *                   4-bit board address
 *
 *
 *  RETURN:          (int)
 *                   NORMAL:  normal return
 *
 *
 *  FUNCTIONS
 *  CALLED:          pi_gets (),
 *                   mem_rd (),
 *                   mem_wr (),
 *                   pi_cvt (),
 *                   pi_rdfil (),
 *                   pi_wrfil ()
 *
 *
 *
 *  AUTHOR:          Copyright 1989
 *                   Durwin D. Nigus
 *
 *
 *  DATE CREATED:    24 July 1989          Version 1.00
 *
 *  REVISIONS:       None.
 *
```

```
 *
***********************************************************/
#include <stdio.h>
#include <ctype.h>
#include <math.h>

#include <stdlib.h>

#include "pi_ddn.h"

int   fc_das (board)

int   board;
(
    char        prompt[PRMPT_MAX];

    int         error, i, num_points, value;

    unsigned    addr, cdc;

    double      freq;


      printf ("ON-BOARD FILTER CONFIGURATION . . . \n\n"
            "C)  change CS7008 sampling frequency \n"
            "L)  load filter configuration from disk \n"
            "M)  make new filter configuration file \n"
            "V)  view filter's coefficient memory \n"
            "\n\n");

    pi_gets (PRMPT_MAX, prompt);

    switch (tolower (prompt[0]))
        {
        case 'c':  /*  change clock divider value   */

/*          -------------------------------------------*/
/*          Display filter sampling frequency.         */
/*          -------------------------------------------*/
            cdc = mem_rd(board, FIL_CDC, REGULAR, FILTER);

            freq = FIL_XTL / 6.0 / pow(2.0, (double)cdc)
                   / 1000.0;
            printf ("\n\n"
                    "The present sampling frequency is "
```

```
                    "%f kHz \n\n"
                    "Select the new clock division code "
                    "from the following: \n\n"
                    "cdc            samp. freq (kHz) \n"
                    "----------------------------- \n",
                     freq);
            for (i = 1 ; i < 8 ; i++)
                {
                freq = FIL_XTL / 6.0 /
                      pow (2.0, (double)i) / 1000.0;
                printf ("%i            %3.1f \n", i, freq);
                }
            printf ("\n\n"
                    "The new cdc value is ");
            pi_gets (PRMPT_MAX, prompt);
            cdc = pi_cvt (prompt);

/*          -------------------------------------------*/
/*          Check for illegal cdc value (must be       */
/*          bounded by 1 thru 7); if legal, store      */
/*          in coefficent memory.                      */
/*          -------------------------------------------*/
            if (cdc < 0 || cdc > 7)
                printf ("Illegal value.  No modifications"
                        " made. \n");
            else
                mem_wr (board, FIL_CDC, (unsigned)i,
                       NORMAL, FILTER);
            break;

        case '1':
/*          -------------------------------------------*/
/*          Load coefficients from disk.               */
/*          -------------------------------------------*/

/*          -------------------------------------------*/
/*          Data buffer will be over-written.  Examine */
/*          the buffer-size and warn the user if       */
/*          necessary.                                 */
/*          -------------------------------------------*/
            if (pi.fix[DATA_LEN] != 0)
                {
```

```
                printf ("\n\n"
                        "Data buffer is NOT empty.  This"
                        " routine will purge \n"
                        "buffer contents.  Proceed? (y/n)"
                        "  ");

                pi_gets (PRMPT_MAX, prompt);

                if (tolower (prompt[0]) != 'y')
                    {
                    printf ("\n\n"
                            "Routine abandoned. \n");
                    return (ERR);
                    }
                else
                    printf ("\n\n");
                }
/*          ----------------------------------------------*/
/*          Read the data-file from disk.                 */
/*          ----------------------------------------------*/
            if (pi_rdfil())
                {
                printf ("\n\n"
                        "Error during file read. \n");
                return (NORMAL);
                }

/*          ----------------------------------------------*/
/*          Store the retrieved coefficients in the       */
/*          filter's memory -- starting at memory         */
/*          address 0.                                    */
/*          ----------------------------------------------*/
            printf ("\n"
                    "Values are being written "
                    "to filter's memory . . . ");

            data_dir (BUS_WRITE, board, FILTER);
            for (addr = 0 ; addr < 64 ; addr++)
                {
                mem_wr (board, addr, pi_data[addr], FAST,
                        FILTER);
                }
            printf ("\n"
                    "Coefficient storage being "
                    "tested . . . \n\n");
```

```
            error = 0;
            for (addr = 0 ; addr < 64 ; addr++)
                {
                i = mem_rd (board, addr, FAST, FILTER);
                if (i != pi_data[addr])
                    {
                    printf ("    Error!    "
                            "Addr=%x h, stored=%x h, "
                            "should be=%x h \n", addr,
                             i, pi_data[addr]);
                    error++;
                    }
                }
            if (error == 0)
                printf ("No errors reported. \n");
            else
                printf ("Filter coefficients NOT "
                        "installed properly. \n");

/*          Zero the data-buffer length.                     */
            pi.fix[DATA_LEN] = 0;

            break;


        case 'm':
/*        ----------------------------------------------*/
/*        Make new filter configuration file.          */
/*        ----------------------------------------------*/
          if (pi.fix[DATA_LEN] != 0)
               {
               printf ("\n\n"
                       "Data buffer is NOT empty.  This"
                       " routine will purge \n"
                       "buffer contents.  Proceed? (y/n)"
                       "  ");

               pi_gets (PRMPT_MAX, prompt);

               if (tolower (prompt[0]) != 'y')
                   {
                   printf ("\n\n"
                           "Routine abandoned. \n");
                   return (NORMAL);
```

```
                )
            }

        printf ("\n\n"
                "This is a very primative data-entry"
                " routine which \n"
                "enables the entry of CS7008 coeff"
                "icients into a disk file. \n\n"
                "Do you wish to continue this "
                "routine?"
                " (y/n) ");

        pi_gets (PRMPT_MAX, prompt);

        if (tolower (prompt[0]) != 'y')
            {
            printf ("Routine abandoned. \n");
            return (NORMAL);
            }

        printf ("\n\n"
            "Instructions:  Type the coefficient "
            "for each address. \n"
            "Unfortunately, this coefficient-"
            "entry routine does not permit \n"
            "error correction. \n\n"
            "If you make an error during entry, "
            "you must make changes to the file  \n"
            "with a text editor.  Sorry. \n\n"
            "You may stop the data-entry by "
            "typing a (-1) \n\n");
        for (addr = 0; addr < 64 ; addr++)
            {
            printf ("\n"
                    "addr = %2x h, coeff = ", addr);
            pi_gets (PRMPT_MAX, prompt);
            i = pi_cvt (prompt);
```

```
/*                  -----------------------------------------*/
/*                  Abandon entry???                         */
/*                  -----------------------------------------*/
                    if (i == -1)
                        {
                        printf ("\n\n"
                                "Routine abandoned. \n\n");
                        return (NORMAL);
                        }

                    pi_data[addr] = i;
                    }

/*              ---------------------------------------------*/
/*              Assign buffer length value                   */
/*              ---------------------------------------------*/
                pi.fix[DATA_LEN] = 64;


/*              =============================================*/
/*              Save the data to a disk file.                */
/*              ---------------------------------------------*/
                pi_wrfil ();

                printf ("\n\n"
                        "File has been saved.  To install "
                        "this set of coefficients in the \n"
                        "filter, use the 'load filter config"
                        "uration' routine listed "
                        "earlier. \n");
                break;


            case 'v':
/*              ---------------------------------------------*/
/*              View filter coefficients.                    */
/*              ---------------------------------------------*/
                printf ("\n\n"
                        "View filter coefficents routine.  "
                        "Use CTRL-S to pause display. \n\n");
```

```
/*          ---------------------------------------------*/
/*          Set the data direction then retrieve each    */
/*          of the coefficent values.                    */
/*          ---------------------------------------------*/
            data_dir (BUS_READ, board, FILTER);
            for (addr = 0 ; addr < 64 ; addr++)
                {
                i = mem_rd (board, addr, FAST, FILTER);
                printf ("\n"
                        "addr=%x h, coeff=%x h ",addr, i);
                )

            break;
        }
    return (0);
}
```

PCPI-DAS SOURCE CODE: SYSTEM COMMAND FUNCTIONS

```
/************************************************************
 *
 *   SOURCE:          fi_das.c
 *
 *
 *   FUNCTION:        fi_das (board)
 *
 *
 *   DESCRIPTION:     The purpose of this function is to
 *                    simulate the "filter control"
 *                    command issued to the DAS.
 *
 *
 *   DOCUMENTATION
 *   FILES:           None.
 *
 *
 *   ARGUEMENTS:
 *      board         (input) int
 *                    4-bit board address
 *
 *
 *   RETURN:          (int)
 *                    NORMAL:  normal return
 *
 *
 *   FUNCTIONS
 *   CALLED:          ctrl_wr (),
 *                    pi_gets ()
 *
 *
 *   AUTHOR:          Copyright 1989
 *                    Durwin D. Nigus
 *
 *
 *   DATE CREATED:    29 May 1989          Version 1.00
 *
 *   REVISIONS:       None.
 *
 *
************************************************************/
```

```
#include <stdio.h>
#include <ctype.h>

#include "pi_ddn.h"

int    fi_das (board)

int    board;
{
      char prompt[PRMPT_MAX];


    printf ("ON-BOARD FILTER SELECTION . . . \n\n"
            "I)   filter in-circuit \n"
            "O)   filter out of circuit \n"
            "\n\n\n");

    pi_gets (PRMPT_MAX, prompt);

    switch (tolower (prompt[0]))
        {
        case 'i':
            ctrl_wr (board, FIL_EN, FILTER_IN);
            break;

         case 'o':
            ctrl_wr (board, FIL_EN, FILTER_OUT);
            break;

         default:
            printf ("\n\n No action taken.");
            break;
        }

    return (NORMAL);
}
```

```
/************************************************************
 *
 *   SOURCE:          fs_das.c
 *
 *
 *   FUNCTION:        fs_das (board)
 *
 *
 *   DESCRIPTION:     The purpose of this function is to
 *                    simulate the "full scale signal range"
 *                    command issued to the DAS.
 *
 *
 *   DOCUMENTATION
 *   FILES:           None.
 *
 *
 *   ARGUEMENTS:
 *      board         (input) int
 *                    4-bit board address
 *
 *
 *   RETURN:          (int)
 *                    NORMAL:  normal return
 *
 *
 *   FUNCTIONS
 *   CALLED:          ctrl_wr (),
 *                    pi_gets ()
 *
 *
 *   AUTHOR:          Copyright 1989
 *                    Durwin D. Nigus
 *
 *
 *   DATE CREATED:    29 May 1989          Version 1.00
 *
 *   REVISIONS:       None.
 *
 *
************************************************************/
```

```
#include <stdio.h>
#include <ctype.h>
#include <stdlib.h>

#include "pi_ddn.h"

int   fs_das (board)

int  board;
{
      char prompt[PRMPT_MAX];

      int  menu;

/*==========================================================*/
/*   Gain set for A/D board.                                */
/*----------------------------------------------------------*/
    printf ("GAIN SET . . . for board %i\n\n"
            "1)   gain = 1        5 V  FS \n"
            "10)  gain = 10     500 mV FS \n"
            "100) gain = 100     50 mV FS \n"
            "200) gain = 200     25 mV FS \n"
            "500) gain = 500     10 mV FS \n"
            "\n\n\n", board);

     pi_gets (PRMPT_MAX, prompt);
     menu = atoi (prompt);

     switch (menu)
        {
        case 1:
            ctrl_wr (board, GAIN_SEL, GAIN_1);
            break;

         case 10:
            ctrl_wr (board, GAIN_SEL, GAIN_10);
            break;

         case 100:
            ctrl_wr (board, GAIN_SEL, GAIN_100);
            break;

         case 200:
            ctrl_wr (board, GAIN_SEL, GAIN_200);
            break;
```

```
         case 500:
            ctrl_wr (board, GAIN_SEL, GAIN_500);
            break;

         default:
            printf ("\n\n No action taken.");
            break;
        }

    return (NORMAL);
}
```

PCPI-DAS SOURCE CODE: SYSTEM COMMAND FUNCTIONS

```
/**********************************************************
 *
 *  SOURCE:          gs_das.c
 *
 *
 *  FUNCTION:        gs_das (board)
 *
 *
 *  DESCRIPTION:     The purpose of this function is to
 *                   present the status of the A/D board on
 *                   screen -- a simulation of the command
 *                   "get status."
 *
 *
 *  DOCUMENTATION
 *  FILES:           None.
 *
 *
 *  ARGUEMENTS:
 *
 *     board         (input) int
 *                   4-bit board address
 *
 *
 *  RETURN:          (int)
 *                   NORMAL:  normal return.
 *
 *
 *  FUNCTIONS
 *  CALLED:          timer_rd (),
 *                   give_val (),
 *                   gv_das (),
 *                   rd_bd (),
 *                   rd_bd_wd ()
 *
 *
 *  AUTHOR:          Copyright 1989
 *                   Durwin D. Nigus
 *
 *
 *  DATE CREATED:    29 May 1989          Version 1.00
 *
 *  REVISIONS:       None.
 *
 *
```

```
*************************************************************/
#include <stdio.h>
#include <ctype.h>
#include <stdlib.h>
#include <process.h>

#include "pi_ddn.h"

int     gs_das (board)

int     board;
{
    int         c1, c2, condition, i, i2, sts, trig_lvl;

    unsigned  onesht, period, pt_count;

    float     t_lvl, v_fs;


/*  ======================================================*/
/*  Retrieve the control and status registers.          */
/*  ------------------------------------------------------*/

/* ----- retrieve CONTROL #1  -------------------------*/
    c1  = (int)bd_rd (board, CTRL1);

/* ----- retrieve CONTROL #2  -------------------------*/
    c2  = (int)bd_rd (board, CTRL2);

/* ----- retrieve STATUS REG. --------------------------*/
    sts = (int)bd_rd (board, STATUS);


/*  ======================================================*/
/*  Display the results.                                 */
/*  ------------------------------------------------------*/
    system ("cls");

    printf ("\n\n"
            "A/D Board Status Query. \n\n");

    if (give_val (BD_ER, sts) == BD_ER_ACT)
        printf ("*** BOARD ERROR *** \n\n");

    printf ("Acquisition mode:    ");
```

```
switch (give_val (BD_MODE, c1))
    {
    case STBY:
        printf ("STANDBY \n");
        break;

    case CONV_NOW:
        if (give_val (SAMP_SER_END, sts) ==
                                   SAMP_END_ACT)
            printf ("series completed.\n");
        else
            printf ("acq in progress.\n");

        break;

    case CONV_TR:
        printf ("conv on trig, ");
        if (give_val (SAMP_SER_END, sts) ==
                                   SAMP_END_ACT)
            printf ("series completed.\n");
        else
            {
            if (give_val (TRIG_RECVD, sts) ==
                                       TR_REC_ACT)
                printf ("acq in progress.\n");
            else
                printf ("waiting for trigger.\n");
            }
        break;

    case CONV_TR_PRE:
        printf ("conv pre-trig, ");
        if (give_val (SAMP_SER_END, sts) ==
                                   SAMP_END_ACT)
            printf ("series completed.\n");
        else
            {
            if (give_val (TRIG_RECVD, sts) ==
                                        TR_REC_ACT)
                printf ("acq in progress.\n");
            else
                printf ("waiting for trigger.\n");
            }
        break;
```

```
        }

    printf ("Trigger source (if selected) is:  ");
    switch (give_val (TRIG_SEL, c1))
        {
        case TR_SIG:
            printf ("signal\n");
            break;

        case TR_SIG_BUS:
            printf ("signal with bus-trigger control\n");
            break;

        case TR_PANEL:
            printf ("A/D board front panel tigger "
                    "input \n");
            break;

        case TR_BUS:
            printf ("system bus \n");
            break;
        }

    printf ("Trigger edge:  ");

    if (give_val (TRIG_EDGE, c1) == TR_EDGE_POS)
        printf ("rising edge\n");
    else
        printf ("falling edge\n");

    printf ("Clock source:  ");

    switch (give_val (CLOCK_SEL, c1))
        {
        case CLK_INT_HI:
            printf ("internal, high-speed  \n");
            break;

        case CLK_INT_LO:
            printf ("internal, low-speed  \n");
            break;

        case CLK_BUS:
            printf ("bus clock \n");
            break;
```

```
        case CLK_PANEL:
            printf ("A/D board front panel  \n");
            break;
        }

    printf ("Signal input:  ");
    switch (give_val (SIG_SEL, c2))
        {
        case SIG_REMOVE:
            printf ("no signal selected, "
                    "shorted inputs\n");
            break;

        case SIG_APPLIED:
            printf ("signal connected  \n");
            break;

        case SIG_REF:
            printf ("internal reference  \n");
            break;

        case SIG_TR_LVL:
            printf ("trigger level \n");
            break;
        }

    printf ("Input overload protection:  ");
    if (give_val (PROTECT, c2) == PROTECT_ON)
        printf ("ENABLED. \n");
    else
        printf ("CAUTION!  Protection disabled.\n");

    printf ("On-board filter status:  ");
    if (give_val (FIL_EN, c1) == FILTER_IN)
        printf ("filter in-circuit. \n");
    else
        printf ("filter out of circuit. \n");

    printf ("Gain selection:  ");
```

```
    switch (give_val (GAIN_SEL, c2))
        {
        case GAIN_1:
        printf ("gain = 1, 5 V FS\n");
        v_fs = 5.0;
        break;

        case GAIN_10:
        printf ("gain = 10, 500 mV FS  \n");
        v_fs = 5.0 / 10.0;
        break;

        case GAIN_100:
        printf ("gain = 100, 50 mV FS  \n");
        v_fs = 5.0 / 100.0;
        break;

        case GAIN_200:
        printf ("gain = 200, 25 mV FS  \n");
        v_fs = 5.0 / 200.0;
        break;

        case GAIN_500:
        printf ("gain = 500, 10 mV FS  \n");
        v_fs = 5.0 / 500.0;
        break;

        default:
        printf ("gain selection error . . . \n");
        v_fs = 5.0;
        break;
        }

/*  If in standby mode, report the trigger-level.        */
    if (give_val (BD_MODE, c1) == STBY)
       {
       printf ("\n"
               "Signal trigger level = ");

       ctrl_wr (board, SIG_SEL, SIG_TR_LVL);
```

```
/*      -------------------------------------------------*/
/*      Wait for the relays and the the circuit to       */
/*      arrive at a steady state operating condition.    */
/*      -------------------------------------------------*/
        for (i = 0 ; i < 1000 ; i++)
            {
            i2 = i;
            }

/*      -------------------------------------------------*/
/*      Initiate the conversion by WRITING (value is     */
/*      NOT important) to the AD_DIRECT register.        */
/*      -------------------------------------------------*/
        bd_wr (board, AD_DIRECT, 0, REGULAR);

/*      -------------------------------------------------*/
/*      When the end-of-conversion status is active,     */
/*      retrieve the A/D data.                           */
/*      -------------------------------------------------*/
        condition = EOC_ACT + 1;

        while (condition != EOC_ACT)
            {
/*          ---------------------------------------------*/
/*          Retrieve STATUS REG. and check eoc status.   */
/*          ---------------------------------------------*/
            condition = give_val (EOC,
                    (int)bd_rd (board, STATUS));
            }

/*      -------------------------------------------------*/
/*      Retrieve data directly from A/D converter.       */
/*      -------------------------------------------------*/
        trig_lvl = (int)bd_rd (board, AD_DIRECT);

        t_lvl = (float)trig_lvl * v_fs * 2.0 / 65535.0;
        printf ("%1.5f volts.", t_lvl);

/*      Reset the signal source to NONE.                 */
        ctrl_wr (board, SIG_SEL, SIG_REMOVE);
        }

    printf ("\n\n");
    return (NORMAL);
}
```

```
/************************************************************
 *
 *  SOURCE:         gv_das.c
 *
 *
 *  FUNCTION:       gv_das (board)
 *
 *
 *  DESCRIPTION:    The purpose of this function is to
 *                  first select a signal source,
 *                  initiate conversion, then retrieve
 *                  the converted value and return it
 *                  to the calling function.
 *
 *
 *  DOCUMENTATION
 *  FILES:          None.
 *
 *
 *  ARGUEMENTS:
 *
 *     board        (input) int
 *                  4-bit board address
 *
 *
 *  RETURN:         int
 *                  0       if improper option selected
 *                          during menu selection;
 *                  value   if conversion occurs.
 *
 *
 *  FUNCTIONS
 *  CALLED:         bd_rd (),
 *                  bd_wr (),
 *                  ctrl_wr (),
 *                  pi_gets ()
 *
 *
 *  AUTHOR:         Copyright 1989
 *                  Durwin D. Nigus
 *
 *
 *  DATE CREATED:   1 June 1989         Version 1.00
 *
 *  REVISIONS:      None.
```

```
 *
 *
***********************************************************/
#include <stdio.h>
#include <string.h>
#include <ctype.h>

#include "pi_ddn.h"

int   gv_das (board)

int   board;
{
      char    prompt[PRMPT_MAX], selection[80];

      int     condition, source_ctrl;


/*   ----------------------------------------------------*/
/*   Select the signal source.                           */
/*   ----------------------------------------------------*/
     printf ("Select signal source from the following \n\n"
             "E)  External signal \n"
             "T)  Trigger level \n"
             "R)  Reference level \n"
             "S)  Shorted inputs \n\n"
             "selection ? ");

     pi_gets (PRMPT_MAX, prompt);


     switch (tolower (prompt[0]))
         {
         case 'e':
             source_ctrl = SIG_APPLIED;
             strcpy (selection, "Front panel signal");
             break;


         case 't':
             source_ctrl = SIG_TR_LVL;
             strcpy (selection, "Signal trigger level");
             break;
```

```
        case 'r':
            source_ctrl = SIG_REF;
            strcpy (selection, "Internal reference");
            break;


        case 's':
            source_ctrl = SIG_REMOVE;
            strcpy (selection, "normal:  inputs shorted");
            break;

        default:
            printf ("\n\n"
                    "Invalid signal source. \n");
            return (0);
            break;
        }

    printf ("\n\n"
            "Source selected was:  %s\n", selection);

/*  ----------------------------------------------------*/
/*  Adjust the value of the appropriate control reg-    */
/*  ister to select the appropriate signal source.      */
/*  ----------------------------------------------------*/
    ctrl_wr (board, SIG_SEL, source_ctrl);

/*  ----------------------------------------------------*/
/*  Wait for the relays and the rst of the circuit to   */
/*  arrive at a steady state operating condition.       */
/*                                                      */
/*  At the very least, one must wait 20 msec for the    */
/*  signal routingrelays to engage.                     */
/*                                                      */
/*  In this test routine, the delay has been skipped    */
/*  and a "conversion prompt" is used to ensure         */
/*  enough time elapses between relay set-up and the    */
/*  A/D conversion.                                     */
/*  ----------------------------------------------------*/
    printf ("\n\n"
            "type   <enter>  to convert. ");
    pi_gets (PRMPT_MAX, prompt);
    printf ("\n\n");

/*  ----------------------------------------------------*/
```

```
/*  Initiate the conversion by WRITING (value is          */
/*  NOT important) to the AD_DIRECT register.             */
/*  -----------------------------------------------------*/
    bd_wr (board, AD_DIRECT, 0, REGULAR);

/*  -----------------------------------------------------*/
/*  When the end-of-conversion status is active,          */
/*  retrieve the A/D data.                                */
/*  -----------------------------------------------------*/
    printf ("\n waiting ");
    condition = EOC_ACT + 1;

    while (condition != EOC_ACT)
        {
/*        -----------------------------------------------*/
/*        Retrieve STATUS REG. and check eoc status.      */
/*        -----------------------------------------------*/
          condition = give_val (EOC,
                      (int)bd_rd (board, STATUS));
          printf (" *");
          }

/*  -----------------------------------------------------*/
/*  Retrieve the data directly from the A/D converter     */
/*  -----------------------------------------------------*/
    printf ("\n");
    return ((int)bd_rd (board, AD_DIRECT));
}
```

```
/***********************************************************
 *
 *   SOURCE:          mt_das.c
 *
 *
 *   FUNCTION:        mt_das (board)
 *
 *
 *   DESCRIPTION:     The purpose of this function is to
 *                    test the memory on the A/D board.
 *                    This routine uses a "marching 1s"
 *                    technique over a specified region
 *                    of memory.
 *
 *
 *   DOCUMENTATION
 *   FILES:           None.
 *
 *
 *   ARGUEMENTS:
 *      board         (input) int
 *                    4-bit board address
 *
 *
 *   RETURN:          (int)
 *                    NORMAL:  normal return
 *
 *
 *   FUNCTIONS
 *   CALLED:          bd_rd (),
 *                    give_val (),
 *                    mem_rd (),
 *                    mem_wr (),
 *                    pi_cvt (),
 *                    pi_gets ()
 *
 *
 *
 *   AUTHOR:          Copyright 1989
 *                    Durwin D. Nigus
 *
 *
 *   DATE CREATED:    29 May 1989          Version 1.00
 *
 *
```

```
 *  REVISIONS:      4Aug89     Memory test for on-board
 *                             filter removed.
 *                  4Sep89     Memory display for EPROM
 *                             and FILTER added.
 *
 *
************************************************************/
#include <stdio.h>
#include <ctype.h>
#include <stdlib.h>

#include "pi_ddn.h"

int   mt_das (board)

int   board;
{

    char     prompt[PRMPT_MAX];

    int      c1, dest, menu;

    unsigned addr_st,
             addr_end,
             addr_pres,
             errors,
             max_num = 0xffff,   /*  64K memory size    */
             num_test,
             temp_val,
             test_1 = 0x5555,    /*  alternating ones   */
             test_val;

/*  -----------------------------------------------------*/
/*  Board is in standby mode.  Select the operation     */
/*  of this function from a menu.                        */
/*  -----------------------------------------------------*/
    printf ("MEMORY TEST . . . \n\n"
            "S)  sample memory (64K x 16 bit) \n"
            "E)  display EPROM values \n"
            "F)  display FITLER values \n"
            "Q)  quit this routine \n\n"
            "selection ? ");

    pi_gets (PRMPT_MAX, prompt);
```

```
    printf ("\n\n");

    switch (tolower (prompt[0]))
        {
        case 's':
/*          -------------------------------------------*/
/*          Test on-board sample memory.               */
/*          -------------------------------------------*/

/*          -------------------------------------------*/
/*          Verify board is in standby mode.  Memory   */
/*          operations may not occur while board is in */
/*          non-standby mode.                          */
/*          -------------------------------------------*/
            c1  = (int)bd_rd (board, CTRL1);
            if (give_val (BD_MODE, c1) != STBY)
                {
                printf ("Board is in non-standby mode."
                        "\n\n"
                        "Memory test may only occur while"
                        " in standby mode. \n\n");
                return (ERR);
                }

            printf ("Testing ON BOARD MEMORY. \n");

/*          Prompt for test parameters.                */
            printf ("\n"
                    "Number of consecutive memory tests"
                    " [max = %u (dec)] ", max_num);

            pi_gets (PRMPT_MAX, prompt);
            num_test = (unsigned)pi_cvt (prompt);
            printf ("\n\n");

            if (num_test == 0)
                {
                printf ("Memory test terminated. \n");
                return (ERR);
                }

            printf ("Starting address???  ");
            pi_gets (PRMPT_MAX, prompt);
            addr_st = (unsigned)pi_cvt (prompt);
            printf ("\n\n");
```

```
            addr_end = addr_st + num_test;

            printf ("\n\n"
                    "Starting address = %x hex, \n"
                    "ending   address = %x hex. \n\n"
                    "The test has begun. \n",
                    addr_st, addr_end);

/*          --------------------------------------------*/
/*          Set the bus data direction, then            */
/*          write the value to the memory positions.    */
/*          --------------------------------------------*/
            data_dir (BUS_WRITE, board, ON_BD_MEM);
            addr_pres = addr_st;
            test_val = test_1;

            while (addr_pres  != addr_end)
                (
                mem_wr (board, addr_pres, test_val,
                        FAST, ON_BD_MEM);

/*              Toggle the test value.                  */
                test_val = ~test_val;

                addr_pres++;
                }

/*  ----------------------------------------------------*/
/*  The value has been written to the desired source.   */
/*  Now, retrieve the contests of each of these memory  */
/*  positions and compare them with the value sent.     */
/*                                                      */
/*  If an error is detected, report it to the user.     */
/*  ----------------------------------------------------*/
            data_dir (BUS_READ, board, ON_BD_MEM);
            errors = 0;
            test_val = test_1;
            addr_pres = addr_st;

            while (addr_pres != addr_end)
                (
                temp_val = mem_rd (board, addr_pres,
                                   FAST, ON_BD_MEM);
```

```
                    if (temp_val != test_val)
                        {
                        errors++;
                        printf ("%u) Error!  addr = %x: write"
                          " = %x, read = %x \n", errors,
                           addr_pres, test_val, temp_val);
                        )

/*                  Toggle the test value.                       */
                    test_val = ~test_val;

                    addr_pres++;
                    )

                printf ("\n\n"
                  "Memory test completed, %u errors reported."
                  " \n", errors);
                break;

            case 'e':
/*              ------------------------------------------*/
/*              Display contents of EPROM (for test only)   */
/*              ------------------------------------------*/
                printf ("\n\n"
                        "Display EPROM contents. \n\n");

                printf ("Starting address:  ");
                pi_gets (PRMPT_MAX, prompt);
                addr_st = (unsigned)pi_cvt (prompt);
                addr_st = (addr_st > 8192) ?
                          (8192) : (addr_st);

                printf ("\n\n"
                        "Ending address:  ");
                pi_gets (PRMPT_MAX, prompt);
                addr_end = pi_cvt (prompt);
                addr_end = (addr_end > 8192) ?
                           (8192) : (addr_end);

                printf ("\n\n");
```

```
            if (addr_st > addr_end)
                {
                printf ("EPROM display terminated. \n");
                return (ERR);
                }

            printf ("Starting address = %x hex, \n"
                    "ending   address = %x hex. \n\n",
                    addr_st, addr_end);


            addr_pres = addr_st;

            for (addr_pres = addr_st    ;
                 addr_pres <= addr_end ;
                 addr_pres++             )
                {
                temp_val = mem_rd (board, addr_pres,
                                   FAST, EPROM);

                printf ("Addr = %4x h, contents = %2xh"
                        "\n", addr_pres, temp_val);
                }

            break;

        case 'f':
/*          -------------------------------------------*/
/*          Display contents of FILTER (for test only)  */
/*          -------------------------------------------*/
            printf ("\n\n"
                    "Display FILTER contents. \n\n");


            printf ("Starting address (0-63)???  ");
            pi_gets (PRMPT_MAX, prompt);
            addr_st = (unsigned)pi_cvt (prompt);
            addr_st = (addr_st > 64) ?
                      (64) : (addr_st);
```

```
printf ("Ending address (0-63)???  ");
pi_gets (PRMPT_MAX, prompt);
addr_end = (unsigned)pi_cvt (prompt);

addr_end = (addr_end > 64) ?
           (64) : (addr_end);


if (addr_st > addr_end)
    {
    printf ("FILTER display terminated. \n");
    return (ERR);
    }


printf ("\n\n"
        "Starting address = %2x h, \n"
        "ending   address = %2x h. \n\n",
        addr_st, addr_end);


data_dir (BUS_READ, board, FILTER);

for (addr_pres = addr_st    ;
     addr_pres <= addr_end  ;
     addr_pres++             )
    {
    temp_val = mem_rd (board, addr_pres,
                       FAST, FILTER);
    printf ("Addr = %2x h, contents = %2xh"
            ", (%3u dec)\n", addr_pres,
            temp_val, temp_val);
    }

break;
```

```
        case 'q':
        default:
/*              --------------------------------------------*/
/*              Quit or abnormal termination.               */
/*              --------------------------------------------*/
                printf ("Memory test terminated. \n");
                return (ERR);

        }

    printf ("\n\n");

    return (NORMAL);
}
```

```
/************************************************************
 *
 *  SOURCE:          rs_das.c
 *
 *
 *  FUNCTION:        rs_das (board)
 *
 *
 *  DESCRIPTION:     The purpose of this function is to
 *                   simulate the "retrieve samples"
 *                   command issued to the DAS.
 *
 *
 *  DOCUMENTATION
 *  FILES:           None.
 *
 *
 *  ARGUEMENTS:
 *
 *     board         (input) int
 *                   4-bit board address
 *
 *
 *
 *  RETURN:          (int)
 *                   NORMAL:  normal return
 *                   ERR   :  abnormal return
 *
 *
 *  FUNCTIONS
 *  CALLED:          bd_rd (),
 *                   give_val (),
 *                   pi_cvt (),
 *                   pi_gets (),
 *                   pi_wrfil ()
 *
 *
 *  AUTHOR:          Copyright 1989
 *                   Durwin D. Nigus
 *
 *
 *  DATE CREATED:    29 May 1989          Version 1.00
 *
 *  REVISIONS:       None.
 *
```

```
 *
***********************************************************************/
#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>

#include "pi_ddn.h"

int   rs_das (board)

int   board;

{
    char     prompt[PRMPT_MAX];

    int      c1, i, sts;

    unsigned  addr,        end_addr,    nth_samp, num_samps,
              start_addr, temp_samps, trig_addr;

/*  ---------------------------------------------------------*/
/*  Verify samples have been taken by the board.            */
/*  ---------------------------------------------------------*/
    if (num_pre_trig == 0 && num_post_trig == 0)
        {
        printf ("No samples have been taken. \n\n");
        return (NORMAL);
        }


/*  =========================================================*/
/*  Retrieve the control and status register.               */
/*  ---------------------------------------------------------*/

/* ----- retrieve CONTROL #1  ---------------------------*/
    c1  = (int)bd_rd (board, CTRL1);

/* ----- retrieve STATUS REG. ----------------------------*/
    sts = (int)bd_rd (board, STATUS);
```

```
/*   ==================================================*/
/*   Examine the control and status values.            */
/*       If board error is present, give option to     */
/*       abandon retrieval.                            */
/*   --------------------------------------------------*/
     if (give_val (BD_ER, sts) == BD_ER_ACT)
         {
         printf ("*** BOARD ERROR ***\n\n"
                 "Do you wish to continue with data "
                 "retrieval?  (y/n): ");
         pi_gets (PRMPT_MAX, prompt);

         if (tolower (prompt[0]) != 'y')
             {
             printf ("\n\n");
             return (ERR);
             }
         }

/*   --------------------------------------------------*/
/*   Determine if the board is still acquiring samples */
/*   and, if so, exit this routine.                    */
/*   --------------------------------------------------*/
     if (give_val (BD_MODE, c1) != STBY
         || give_val (SAMP_SER_END, sts) != SAMP_END_ACT)
         {
         printf ("Board has not completed present "
                 "acquisition duties. \n\n"
                 "Note:  Use 'sc' before retrieving data "
                 "from board.\n\n");
         return (ERR);
         }

/*   ==================================================*/
/*   All pre-conditions for data retrieval have been met */
/*                                                     */
/*   The procedure for retrieving the data is as       */
/*   follows:                                          */
/*                                                     */
/*       1)  If conversion mode was "pre-trigger"      */
/*           (num_pre_trig != 0), retrieve the         */
/*           trigger address.                          */
```

```
/*      2)   Set the memory address on the bus to the     */
/*           first sample to be retrieved.                */
/*      3)   Read the data from the board.                */
/*      4)   Write the data to disk.                      */
/*                                                        */
/*  ----------------------------------------------------*/
    if (num_pre_trig == 0)
        {
        start_addr = FIRST_SAMP;
        end_addr   = num_post_trig;
        }
    else
        {
        trig_addr =  bd_rd (board, TRIG_ADDR);

/*      ------------------------------------------------*/
/*      Calculate the starting address based on the     */
/*      trigger address and the condition of the        */
/*      "memory wrap" status.                           */
/*      ------------------------------------------------*/
        if (give_val (PTR_WRAP, sts) == PTR_WR_ACT)
            start_addr = trig_addr - num_pre_trig
                       + FIRST_SAMP;
        else
           start_addr = (trig_addr > num_pre_trig) ?
           (trig_addr - num_pre_trig + FIRST_SAMP) :
           (FIRST_SAMP) ;

        end_addr = trig_addr + num_post_trig;
        )

/*  ----------------------------------------------------*/
/*  Calculate the number of samples to be acquired.     */
/*  ----------------------------------------------------*/
    num_samps = abs (end_addr - start_addr) + 1 ;

/*  ----------------------------------------------------*/
/*  The address bounds have been determined and         */
/*  retrieval may begin.                                */
/*  ----------------------------------------------------*/
    printf ("Sample data retrieval from board %d \n\n"
            "Starting address = %x hex \n"
            "Ending address   = %x hex \n\n"
            "A total of %u samples. \n\n",
             board, start_addr, end_addr, num_samps);
```

```
/*  =====================================================*/
/*  The data retrieval routine.                          */
/*  -----------------------------------------------------*/
    printf ("\n"
            "Retrieve how many samples (%u):  ",
            num_samps);

    if (pi_gets (PRMPT_MAX, prompt) != 0)
        num_samps = (unsigned)pi_cvt (prompt);

    if (num_samps == 0)
        {
        printf ("\n\n"
                "No samples requested.\n");
        return (ERR);
        }


    num_samps = (num_samps > MAX_DATA) ? (MAX_DATA) :
                                         (num_samps);


/*  Pre-assign the nth-sample value.                     */
    nth_samp = 1;
    printf ("\n\n"
            "%u samples to be retrieved. \n\n"
            "Retrieval begins with which sample (%u): ",
             num_samps, nth_samp);

    if (pi_gets (PRMPT_MAX, prompt) != 0)
        nth_samp = (unsigned)pi_cvt (prompt);

/*  -----------------------------------------------------*/
/*  Check for an illegal request . . .                   */
/*  -----------------------------------------------------*/
    if (nth_samp > num_samps || nth_samp <= 0)
        {
        printf ("\n\n"
                "Illegal request. \n\n");
        return (ERR);
        }
```

```
/*  -----------------------------------------------------*/
/*  Assign the number of samples to pi.fix[DATA_LEN]     */
/*  and set the address.                                 */
/*  -----------------------------------------------------*/
    pi.fix[DATA_LEN] = num_samps - 1;
    addr = start_addr + nth_samp ;

/*  -----------------------------------------------------*/
/*  Loop through and retireve the data, saving it in     */
/*  the array.  In order to use the FAST memory read     */
/*  mode, pre-set the bus data direction.                */
/*  -----------------------------------------------------*/
    data_dir (BUS_READ, board, ON_BD_MEM);

    for (i = 0 ; i < pi.fix[DATA_LEN]; i++)
        {
        pi_data[i] = mem_rd(board, addr, FAST, ON_BD_MEM);
        addr++;
        }

/*  -----------------------------------------------------*/
/*  The buffer now contains the data retrieved from the */
/*  board.  Give the user the option to save the buffer */
/*  to a disk-file.                                      */
/*  -----------------------------------------------------*/
    printf ("\n\n"
            "The data has been collected from the A/D "
            "Board memory. \n\n"
            "Now, save data buffer to disk. \n\n");

    pi_wrfil();

    return (NORMAL);
}
```

```
/*************************************************************
 *
 *  SOURCE:          sc_das.c
 *
 *
 *  FUNCTION:        sc_das (board)
 *
 *
 *  DESCRIPTION:     The purpose of this function is to
 *                   simulate the "stop conversion"
 *                   command issued to the DAS.  This
 *                   function removes the signal from
 *                   the input amplifier and resets
 *                   the board to the standby mode.
 *
 *
 *  DOCUMENTATION
 *  FILES:           None.
 *
 *
 *  ARGUEMENTS:
 *     board         (input) int
 *                   4-bit board address
 *
 *
 *  RETURN:          (int)
 *                   NORMAL:  normal return
 *
 *
 *  FUNCTIONS
 *  CALLED:          ctrl_wr ()
 *
 *
 *  AUTHOR:          Copyright 1989
 *                   Durwin D. Nigus
 *
 *
 *  DATE CREATED:    29 May 1989          Version 1.00
 *
 *  REVISIONS:       None.
 *
 *
*************************************************************/
```

PCPI-DAS SOURCE CODE: SYSTEM COMMAND FUNCTIONS

```
#include <stdio.h>

#include "pi_ddn.h"

int   sc_das (board)

int   board;
{
/*  =====================================================*/
/*  Set conversion control to STANDBY and disconnect     */
/*  the signal input.                                    */
/*  -----------------------------------------------------*/
    ctrl_wr (board, SIG_SEL, SIG_REMOVE);
    ctrl_wr (board, BD_MODE, STBY);

    return (NORMAL);
}
```

PCPI-DAS SOURCE CODE: SYSTEM COMMAND FUNCTIONS

```
/***************************************************************
 *
 *   SOURCE:          sr_das.c
 *
 *
 *   FUNCTION:        sr_das (board)
 *
 *
 *   DESCRIPTION:     The purpose of this function is to
 *                    simulate the "set sample rate"
 *                    command issued to the DAS.
 *
 *
 *   DOCUMENTATION
 *   FILES:           None.
 *
 *
 *   ARGUEMENTS:
 *      board         (input) int
 *                    4-bit board address
 *
 *
 *   RETURN:          (int)
 *                    NORMAL:  normal return
 *
 *
 *   FUNCTIONS
 *   CALLED:          ctrl_wr (),
 *                    timer_wr ()
 *
 *
 *   AUTHOR:          Copyright 1989
 *                    Durwin D. Nigus
 *
 *
 *   DATE CREATED:    29 May 1989          Version 1.00
 *
 *   REVISIONS:       None.
 *
 *
***************************************************************/
```

```
#include <stdio.h>
#include <ctype.h>
#include <stdlib.h>

#define SLOW_OSC_DIV  1024.0   /*  2^10                        */

#include "pi_ddn.h"

int   sr_das (board)

int   board;
{
    char      prompt[PRMPT_MAX];

    int       sel;

    unsigned  count;

    float     act_freq, clk_div, des_freq,
              f_low, freq, num;


/*  =====================================================*/
/*  Prompt for desired sampling frequency, and exit     */
/*  if out of bounds.                                    */
/*  -----------------------------------------------------*/
    printf ("Desired sampling frequency is (Hz): ");
    pi_gets (PRMPT_MAX, prompt);
    des_freq = atof (prompt);

    f_low = xtal_freq / SLOW_OSC_DIV / 65535.0;

    if (des_freq < f_low || des_freq > F_HIGH)
        {
        printf ("Illegal frequency selected. \n\n");
        return (ERR);
        }
```

```
/*  ------------------------------------------------------*/
/*  Determine the proper selection for the internal      */
/*  oscillator (HIGH or LOW) and set the control reg.    */
/*  ------------------------------------------------------*/
    sel     = (des_freq > F_MID) ?
              (CLK_INT_HI) : (CLK_INT_LO);

    clk_div = (des_freq > F_MID) ?
              (2.0) : (SLOW_OSC_DIV);

    ctrl_wr (board, CLOCK_SEL, sel);


/*  ======================================================*/
/*  Calculate the value for the delay counter.           */
/*  ------------------------------------------------------*/
    num = xtal_freq / clk_div / des_freq;


/*  ======================================================*/
/*  Determine the closest integer to  num  and send      */
/*  to counter.                                          */
/*  ------------------------------------------------------*/
    count = (unsigned)(num + 0.5);

    timer_wr (board, TMR_SAMP_PD, count);


/*  ======================================================*/
/*  Display the actual frequency set.                    */
/*  ------------------------------------------------------*/
    act_freq = xtal_freq / clk_div / (float)count;

    printf ("\n\n"
            "actual frequency = %10.2f Hz\n"
            "sampling period  = %.7f sec\n\n",
             act_freq, 1.0 / act_freq);

    return (NORMAL);
}
```

PCPI-DAS SOURCE CODE: SYSTEM COMMAND FUNCTIONS

```
/**************************************************************
 *
 *   SOURCE:          ts_das.c
 *
 *
 *   FUNCTION:        ts_das (board)
 *
 *
 *   DESCRIPTION:     The purpose of this function is to
 *                    simulate the "trigger select" command
 *                    issued to the DAS.
 *
 *
 *   DOCUMENTATION
 *   FILES:           None.
 *
 *
 *   ARGUEMENTS:
 *      board         (input) int
 *                    4-bit board address
 *
 *
 *   RETURN:          (int)
 *                    NORMAL:  normal return
 *
 *
 *   FUNCTIONS
 *   CALLED:          ctrl_wr (),
 *                    pi_gets ()
 *
 *
 *   AUTHOR:          Copyright 1989
 *                    Durwin D. Nigus
 *
 *
 *   DATE CREATED:    29 May 1989          Version 1.00
 *
 *   REVISIONS:       None.
 *
 *
**************************************************************/
```

```
#include <stdio.h>
#include <ctype.h>


#include "pi_ddn.h"

int   ts_das (board)

int   board;
{
    char     prompt[PRMPT_MAX];
    int      value,
             wr_ctrl = YES;

/*=========================================================*/
/*  TRIGGER SOURCE SELECTION                               */
/*---------------------------------------------------------*/
    printf ("\n\n"
            "A/D board:  trigger source selector . . . "
            "\n\n"
            "S) signal level  \n"
            "Z) signal level w/bus pull_down  \n"
            "P) front panel trigger input \n"
            "B) bus trigger  \n"
            "\n\n\n");

    pi_gets (PRMPT_MAX, prompt);

    switch (tolower (prompt[0]))
        {
         case 's':
            value = TR_SIG;
            break;

         case 'z':
            value = TR_SIG_BUS;
            break;

         case 'p':
            value = TR_PANEL;
            break;

         case 'b':
            value = TR_BUS;
            break;
```

```
            default:
                wr_ctrl = NO;
                printf ("\n\nNo action taken.");
                break;
            }

        if (wr_ctrl == YES)
            ctrl_wr (board, TRIG_SEL, value);
        else
            return (ERR);

/*  ==================================================*/
/*  TRIGGER EDGE SELECTION                            */
/*  --------------------------------------------------*/
    printf ("\n\n"
            "A/D board: trigger edge selection . . . \n\n"
            "R) rising edge (default)\n"
            "F) falling edge  \n"
            "\n\n"
            "edge selected???  ");

    pi_gets (PRMPT_MAX, prompt);

    switch (tolower (prompt[0]))
        {
        case 'f':
            value = TR_EDGE_NEG;
            break;

        default:
        case 'r':
            value = TR_EDGE_POS;
            break;

        }

    if (wr_ctrl == YES)
        ctrl_wr (board, TRIG_EDGE, value);

    return (NORMAL);
}
```

```
/************************************************************
 *
 *  SOURCE:          byte_brk.c
 *
 *
 *  FUNCTION:        byte_brk (dat, msb, lsb)
 *
 *
 *  DESCRIPTION:     The purpose of this function is to
 *                   convert a 16-bit unsigned nunmber
 *                   into its respective LSB  and  MSB.
 *
 *
 *  DOCUMENTATION
 *  FILES:           None.
 *
 *
 *  ARGUEMENTS:
 *
 *     dat           (input) unsigned
 *                   the 16-bit value
 *
 *     msb           (output) *int
 *                   pointer to the 8-bit MSB
 *
 *     lsb           (output) *int
 *                   pointer to the 8-bit LSB
 *
 *
 *  RETURN:          (int)
 *                   NORMAL
 *
 *  FUNCTIONS
 *  CALLED:          None.
 *
 *
 *  AUTHOR:          Copyright 1989
 *                   Durwin D. Nigus
 *
 *
 *  DATE CREATED:    28 May 1989          Version 1.00
 *
 *  REVISIONS:       None.
 *
 *
```

```
*************************************************************/
#include <stdio.h>
#include <ctype.h>

#include "pi_ddn.h"

int    byte_brk (dat, msb, lsb)

unsigned   dat;
int        *msb, *lsb;
{
    *msb  = dat / 256;
    *lsb  = dat - *msb * 256;

    return (NORMAL);
}
```

```
/*************************************************************
 *
 *  SOURCE:         ctrl_wr.c
 *
 *
 *  FUNCTION:       ctrl_wr (board, data_id, value)
 *
 *
 *  DESCRIPTION:    The purpose of this function is to
 *                  write a control value to a control
 *                  register on the A/D board.
 *
 *
 *  DOCUMENTATION
 *  FILES:          None.
 *
 *
 *  ARGUEMENTS:
 *
 *     board        (input) int
 *                  4-bit A/D board address
 *
 *     data_id      (input) int
 *                  contains information about the
 *                  location of the data to be written.
 *
 *                  See  pi_ddn.h  for a description of
 *                  this parameter.
 *
 *     value        (input) int
 *                  8-bit value to be written to a
 *                  control region (from 1 to 7 bits)
 *
 *
 *  RETURN:         (int)
 *                  NORMAL :  no errors encountered
 *                  ERR    :  improper register specs
 *
 *
 *  FUNCTIONS
 *  CALLED:         bd_rd (),
 *                  bd_wr ()
 *
 *
```

```
 *   AUTHOR:          Copyright 1989
 *                    Durwin D. Nigus
 *
 *
 *   DATE CREATED:    28 May 1989          Version 1.00
 *
 *   REVISIONS:       None.
 *
 *
***********************************************************/
#include <stdio.h>
#include <ctype.h>

#include "pi_ddn.h"

int ctrl_wr (board, data_id, value)

int board, data_id, value;
{
    int     cont_reg,
            mask,
            num_bits,
            num_shifts,
            previous,
            reg,
            result,
            shft_val;

    cont_reg = data_id & 3;


/*  -----------------------------------------------------*/
/*  Determine the control register being addressed.     */
/*  -----------------------------------------------------*/

/*  Double check that a control-value is correct. . .   */

    if (cont_reg != CONTROL1 && cont_reg != CONTROL2)
        return (ERR);

    reg = (cont_reg == CONTROL1) ? (CTRL1) : (CTRL2);
```

```
/*==========================================================*/
/*  How this routine works:                                  */
/*       The control registers have various fields           */
/*  which serve several functions (as defined in             */
/*  the pi_ddn.h header).  The position and size of          */
/*  these fields is passed to this routine in                */
/*  'dat_id', and the value to reside in that                */
/*  field is passed in 'value'.                              */
/*                                                           */
/*  This routine does the following things:                  */
/*       1)   Move the data to be written into the           */
/*            proper position.                               */
/*       2)   Generate a mask-byte in order to clear         */
/*            the value retrieved from the control           */
/*            register.                                      */
/*       3)   Retrieve the pre-modify value from the         */
/*            control register;  place the new field        */
/*            value within this value; write this new        */
/*            value to the control register.                 */
/*----------------------------------------------------------*/

/*  ----------------------------------------------------*/
/*  Move the data into the proper position.            */
/*  ----------------------------------------------------*/

/*  Find out the number of shifts, then shift.         */
    num_shifts = (data_id & (7 * BIT_POS)) / BIT_POS;
    shft_val = value << num_shifts;


/*  ----------------------------------------------------*/
/*  Generate the clearing mask.                         */
/*  Determine number of bits in mask . . .              */
/*  (use a 3-bit mask, 7)                               */
/*  ----------------------------------------------------*/
    num_bits = (data_id & (7 * NUM_BTS)) / NUM_BTS;

/*  Shift the number of bits for the field into the    */
/*  proper position in the mask.                       */
    mask = (0xff << num_bits) & 0xff;

/*  Invert the mask (put ones where zeros where).      */
    mask = ~mask;
```

```
/*  Move ones to appropriate position.                          */
    mask = mask << num_shifts;

/*  Invert the mask so that zeros are in the position           */
/*  of the field being modified.                                */
    mask = (~mask) & 0xff;


/*  ----------------------------------------------------------*/
/*  Retrieve the old-value from the control register.        */
/*  Modify it.  Replace with the new value.                  */
/*  ----------------------------------------------------------*/
    previous = (int)bd_rd (board, reg);

/*  Zero the old field . . .                                    */
    result = previous & mask;

/*  . . . then place the new value in the field.                */
    result = shft_val | result;


/*  ----------------------------------------------------------*/
/*  Save the result of the bit alterations (presently        */
/*  in result) in the appropriate control register           */
/*  (determined earlier).                                     */
/*  ----------------------------------------------------------*/
    bd_wr (board, reg, result, REGULAR);

    return (NORMAL);
}
```

```
/*************************************************************
 *
 *   SOURCE:          give_val.c
 *
 *
 *   FUNCTION:        give_val (data_id, num)
 *
 *
 *   DESCRIPTION:     The purpose of this function is
 *                    decipher the contents of the control
 *                    and status registers -- see data_id
 *                    code definitions in the header file
 *                    pi_ddn.h .
 *
 *
 *   DOCUMENTATION
 *   FILES:           None.
 *
 *
 *   ARGUEMENTS:
 *
 *      data_id       (input) int
 *                    the data identifier for the region
 *                    of data that is to be inspected
 *
 *      num           (input) int
 *                    the 8-bit value retrieved from a
 *                    status or control register
 *
 *
 *   RETURN:          (int)
 *                    the value from the field in question
 *
 *
 *   FUNCTIONS
 *   CALLED:          None.
 *
 *
 *   AUTHOR:          Copyright 1989
 *                    Durwin D. Nigus
 *
 *
 *   DATE CREATED:    28 May 1989        Version 1.00

 *   REVISIONS:       None.
```

```
 *
 *
*************************************************************/
#include <stdio.h>
#include <ctype.h>

#include "pi_ddn.h"

int   give_val (data_id, num)

int   data_id, num;
{
    int     mask,  mask_size,  num_shifts;

/*  ---------------------------------------------------*/
/*  1)  Generate a mask to remove the field-value      */
/*      from  num.                                     */
/*  ---------------------------------------------------*/
    mask_size = (data_id & (7 * NUM_BTS)) / NUM_BTS;

/*  Move the proper number of zeros into  mask.        */
    mask = 0xff << mask_size;

/*  Now make these zeros into ones.                    */
    mask = (~mask) & 0xff;


/*  ---------------------------------------------------*/
/*  2)  Determine the position of the mask bits within */
/*      the data byte.                                 */
/*  ---------------------------------------------------*/
    num_shifts = (data_id & (7 * BIT_POS)) / BIT_POS;

/*  Shift the mask generated earlier by the determined */
/*  number of bits.                                    */
    mask <<= num_shifts;

/*  ---------------------------------------------------*/
/*  3)  Mask  num  with the generated mask and return  */
/*      the value from the field to the user.          */
/*  ---------------------------------------------------*/
    num = (num & mask) >> num_shifts;

    return (num);
}
```

```
/************************************************************
 *
 *  SOURCE:          mem_rd.c
 *
 *
 *  FUNCTION:        mem_rd (board, addr, mode, source)
 *
 *
 *  DESCRIPTION:     The purpose of this function is to
 *                   facilitate reading a memory register
 *                   from the specifed memory source,
 *                   namely the on-board filter or the
 *                   sample memory.
 *
 *                   When the filter memory is read, the
 *                   value retrieved from the filter is
 *                   masked such that # of bits used by
 *                   that particular coefficient.  The
 *                   number of bits used by each coeff-
 *                   icient was obtained from the Address
 *                   Map, p. 9-15, Crystal Data Book.
 *
 *
 *  DOCUMENTATION
 *  FILES:           None.
 *
 *
 *  ARGUEMENTS:
 *
 *     board         (input) int
 *                   4-bit board address
 *
 *     addr          (input) unsigned
 *                   memory address
 *
 *     mode          (input) int
 *                   selects between
 *                       FAST      :  memory is read without
 *                                    resetting data direct-
 *                                    ion and bd/reg addr.
 *
 *                       REGULAR   :  presets data direction
 *                                    and board/register
 *                                    values.
 *
```

```
 *     source        (input) int
 *                   selects between
 *                       ON_BD_MEM :  sample memory; or
 *                       FILTER    :  on-board filter
 *                       EPROM     :  on-board EPROM
 *
 *
 *   RETURN:         (unsigned)
 *                   value read from specified memory
 *                   register
 *
 *   FUNCTIONS
 *   CALLED:         byte_brk (),
 *                   data_dir (),
 *                   bus_rd (),
 *                   rd_strob (),
 *                   bus_wr ()
 *
 *
 *   AUTHOR:         Copyright 1989
 *                   Durwin D. Nigus
 *
 *
 *   DATE CREATED:   28 May 1989     Version 1.00
 *
 *   REVISIONS:      24Jul89   Mask for filter-read added.
 *                    4Sep89   EPROM read option added.
 *
 *
***********************************************************/
#include <stdio.h>
#include <math.h>
#include <ctype.h>

#include "pi_ddn.h"

unsigned   mem_rd (board, addr, mode, source)

int    board, mode, source;

unsigned   addr;

{
    int     lsb,
            msb;
```

```
    unsigned    value;

    static int fil_mask[65] = { 5, 6, 5, 6, 5, 6, 2, 0,
                                5, 6, 5, 6, 5, 6, 0, 0,
                                5, 6, 5, 6, 5, 6, 2, 0,
                                5, 6, 5, 6, 5, 6, 3, 0,
                                5, 6, 5, 6, 5, 6, 2, 0,
                                5, 6, 5, 6, 5, 6, 0, 0,
                                5, 6, 5, 6, 5, 6, 2, 0,
                                5, 6, 5, 6, 5, 6, 6, 5 );


/*  =====================================================*/
/*  If the mode bit is set to FAST, skip the board and  */
/*  register setup, and data direction selection.       */
/*  -----------------------------------------------------*/
    if (mode != FAST)
        data_dir (BUS_READ, board, source);

    switch (source)
        {
        case FILTER:
/*          ---------------------------------------------*/
/*          Set the address lines appropriately . . .   */
/*          The filter's address lines are the LSB      */
/*          of the 16-bit address.                      */
/*          ---------------------------------------------*/
            bus_wr (ADDR_LSB, addr);

/*          ---------------------------------------------*/
/*          Read the data from the appropriate data bus */
/*          registers while the bus read strobe is      */
/*          active.                                     */
/*          ---------------------------------------------*/
            rd_strob (ACTIVE, FAST);

            value = ((FILTER & DATA_MASK) == DATA_LOW) ?
                    (bus_rd (DAT_LSB)) :
                    (bus_rd (DAT_MSB)) ;

            rd_strob (INACTIVE, FAST);
```

```
/*          -------------------------------------------*/
/*          Mask the data from the filter.             */
/*          -------------------------------------------*/
            value = value & (int)(pow(2.0,
                    (double)fil_mask[addr]) - 1);

            break;


        case ON_BD_MEM:
/*          -------------------------------------------*/
/*          Set the address lines appropriately . . .  */
/*          -------------------------------------------*/
            byte_brk (addr, &msb, &lsb);
            bus_wr (ADDR_LSB, lsb);
            bus_wr (ADDR_MSB, msb);

/*-----------------------------------------------------*/
/*  Read the data from the appropriate data bus        */
/*  registers while the bus read strobe is active.     */
/*-----------------------------------------------------*/
            rd_strob (ACTIVE, FAST);

            value = bus_rd (DAT_LSB)
                    + 256 * bus_rd (DAT_MSB);

            rd_strob (INACTIVE, FAST);

            break;


    case EPROM:
/*          -------------------------------------------*/
/*          Set the address lines appropriately . . .  */
/*          -------------------------------------------*/
            byte_brk (addr, &msb, &lsb);
            bus_wr (ADDR_LSB, lsb);
            bus_wr (ADDR_MSB, msb);
```

```
/*-----------------------------------------------------*/
/*  Read the data from the appropriate data bus        */
/*  registers while the bus read strobe is active.     */
/*-----------------------------------------------------*/
            rd_strob (ACTIVE, FAST);

            value = ((EPROM & DATA_MASK) == DATA_LOW) ?
                    (bus_rd (DAT_LSB)) :
                    (bus_rd (DAT_MSB)) ;

            rd_strob (INACTIVE, FAST);

            break;
        }

/*-----------------------------------------------------*/
/*  Return the value to the caller.                    */
/*-----------------------------------------------------*/
    return (value);
}
```

```
/**********************************************************
 *
 *  SOURCE:          mem_wr.c
 *
 *
 *  FUNCTION:        mem_wr (board, addr, dat, mode, dest)
 *
 *
 *  DESCRIPTION:     The purpose of this function is to
 *                   facilitate writing to on-board memory,
 *                   including the sample memory and the
 *                   on-board programmable filter.
 *
 *                   NOTE:  This routine has no effect
 *                   when writing to on-board memory
 *                   while the A/D board is in a
 *                   non-standby mode.
 *
 *
 *  DOCUMENTATION
 *  FILES:           None.
 *
 *
 *  ARGUEMENTS:
 *
 *     board         (input) int
 *                   4-bit board address
 *
 *     addr          (input) unsigned
 *                   memory address (16-bit for sample
 *                   memory, 6-bit for filter)
 *
 *     dat           (input) unsigned
 *                   data value to be written to memory
 *                   (16-bit for sample memory, 6-bit
 *                   for filter)
 *
```

```
 *     mode          (input) int
 *                   selects between normal or fast
 *                   memory writes
 *
 *                       NORMAL :  data direction, board
 *                                 and register address
 *                                 are set prior to
 *                                 writing to memory.
 *
 *                       FAST   :  memory is written to
 *                                 immediately.
 *
 *     dest          (input) int
 *                   selects between:
 *
 *                   ON_BD_MEM    :   sample memory
 *                   FILTER       :   programmable filter
 *
 *                   If neither of these destinations
 *                   is selected, this function does
 *                   nothing.
 *
 *
 *   RETURN:         (int)
 *                   NORMAL
 *
 *
 *   FUNCTIONS
 *   CALLED:         byte_brk (),
 *                   data_dir (),
 *                   bus_wr (),
 *                   wr_strob ()
 *
 *
 *   AUTHOR:         Copyright 1989
 *                   Durwin D. Nigus
 *
 *
 *   DATE CREATED:   28 May 1989          Version 1.00
 *
 *   REVISIONS:      None.
 *
 *
*************************************************************/
```

```
#include <stdio.h>
#include <ctype.h>

#include "pi_ddn.h"

int     mem_wr (board, addr, dat, mode, dest)

int     board, mode, dest;

unsigned   addr, dat;
{
    int       lsb, msb;

    if (dest == ON_BD_MEM || dest == FILTER)
        {

/*      ================================================*/
/*      If mode bit is set to FAST, skip the board and  */
/*      register setup, and data direction selection.   */
/*      ------------------------------------------------*/
        if (mode != FAST)
            data_dir (BUS_WRITE, board, dest);
        }

    switch (dest)
        {
            case FILTER:
/*          --------------------------------------------*/
/*          Set the address lines appropriately.        */
/*          Filter is addressable from the LSB lines.   */
/*          --------------------------------------------*/
            bus_wr (ADDR_LSB, addr);

/*          --------------------------------------------*/
/*          Write the data to the filter.               */
/*          First, determine which data byte the filter */
/*          is addressed from.                          */
/*          --------------------------------------------*/
            if ((FILTER & DATA_MASK) == DATA_LOW)
                bus_wr (DAT_LSB, dat);
            else
                bus_wr (DAT_MSB, dat);

            wr_strob ();
            break;
```

```
        case ON_BD_MEM:
/*          --------------------------------------------*/
/*          Set the address lines appropriately.       */
/*          --------------------------------------------*/
            byte_brk (addr, &msb, &lsb);
            bus_wr (ADDR_LSB, lsb);
            bus_wr (ADDR_MSB, msb);

/*          --------------------------------------------*/
/*          Write data to data bus register.           */
/*          --------------------------------------------*/
            byte_brk (dat, &msb, &lsb);
            bus_wr (DAT_LSB, lsb);
            bus_wr (DAT_MSB, msb);

/*          --------------------------------------------*/
/*          Flash the write-data strobe.               */
/*          --------------------------------------------*/
            wr_strob ();

            break;
        }
    return (NORMAL);
}
```

```
/*************************************************************
 *
 *   SOURCE:         sing_bit.c
 *
 *
 *   FUNCTION:       sing_bit (old_byte, dat_id, bit_ctrl)
 *
 *
 *   DESCRIPTION:    The purpose of this function is to
 *                   adjust a specifed bit in a byte and
 *                   return the new value.  This function
 *                   is used with id values associated with
 *                   BUS-CONTROLS e.g. bus trigger, bus
 *                   write, NOT A/D board control register
 *                   modifications.
 *
 *
 *   DOCUMENTATION
 *   FILES:          None.
 *
 *
 *   ARGUEMENTS:
 *
 *      old_byte     (input) int
 *                   the byte value to which bit manip-
 *                   ulation is desired
 *
 *      dat_id       (input) int
 *                   the position of the bit to be adjusted
 *                   (a value between 0 and 7);
 *                   the value of dat_id when ANDed with
 *                   ACT_POSITION is the active_hi or
 *                   active_lo value.
 *
 *      bit_ctrl     (input) int
 *                   ACTIVE or INACTIVE
 *
 *
 *   RETURN:         (int)
 *                   old_byte with the specified bit
 *                   modified as requested
 *
 *
 *   FUNCTIONS
 *   CALLED:         None.
```

```
 *
 *
 *  AUTHOR:         Copyright 1989
 *                  Durwin D. Nigus
 *
 *
 *  DATE CREATED:   29 May 1989          Version 1.00
 *
 *  REVISIONS:      24Jul89   Clean up.
 *
 *
***********************************************************/
#include <stdio.h>
#include <ctype.h>

#include "pi_ddn.h"

int     sing_bit (old_byte, dat_id, bit_ctrl)

int     old_byte, dat_id, bit_ctrl;
{
    int   mask, new_bit, new_byte;

/*  ====================================================*/
/*  Determine whether bit is active low or high        */
/*  by examining ACT_POSITON in data id.               */
/*  ----------------------------------------------------*/
    if ((dat_id & ACT_POSITION) / ACT_POSITION == ACT_LO)
        new_bit = (bit_ctrl == ACTIVE) ? (0) : (1);
    else
        new_bit = (bit_ctrl == ACTIVE) ? (1) : (0);


/*  ====================================================*/
/*  Generate the bit mask:                              */
/*  shift a "1" into the proper bit position.           */
/*  The bit position is given in the three least sig-   */
/*  nificant bits of  dat_id.                           */
/*  ----------------------------------------------------*/
    mask = 0;
    mask = 1 << (dat_id & 7);
```

```
/*  ==========================================================*/
/*  Determine appropriate routine to manipulate bit.         */
/*  ----------------------------------------------------------*/
    switch (new_bit)
        {
        case 1:     /*   make the bit a "1"     */

            new_byte = mask | old_byte;
            break;

        case 0:     /*   make the bit a "0"     */

            new_byte = ~mask & old_byte;
            break;
        }

    return (new_byte);
}
```

```
/*************************************************************
 *
 *   SOURCE:         timer_rd.c
 *
 *
 *   FUNCTION:       timer_rd (board, data_id)
 *
 *
 *   DESCRIPTION:    The purpose of this function is to
 *                   retrieve the 16-bit counter value
 *                   from the specified register on the
 *                   82C54-2 counter chip (present on the
 *                   A/D board).  A full description of this
 *                   device can be found in the 1986 Intel
 *                   Microprocessor Peripheral Databook, p.
 *                   6-294.
 *
 *
 *   DOCUMENTATION
 *   FILES:          None.
 *
 *
 *   ARGUEMENTS:
 *
 *      board        (input) int
 *                   4-bit board address
 *
 *      data_id      (input) int
 *                   selects between the three counter
 *                   based on their function:
 *
 *                       TMR_SAMP_PD : sample period
 *                                     counter
 *                       TMR_ONESHT  : one-shot
 *                       TMR_SAMP_CNT: sample counter
 *
 *
 *   RETURN:         (unsigned)
 *                   the 16-bit nunmber retrieved from
 *                   the specified counter
 *
 *   FUNCTIONS
 *   CALLED:         bd_rd (),
 *                   bd_wr (),
 *                   ctrl_wr ()
```

```
 *
 *
 *  AUTHOR:          Copyright 1989
 *                   Durwin D. Nigus
 *
 *
 *  DATE CREATED:    28 May 1989      Version 1.00
 *
 *  REVISIONS:       None.
 *
 *
*********************************************************/
#include <stdio.h>
#include <ctype.h>

#include "pi_ddn.h"

#define  CNT_0     2    /*  Values added to the control */
#define  CNT_1     4    /*  codes that enable i/o with  */
#define  CNT_2     8    /*  the respective counter.     */

#define  CNT_STS  0x40  /*  status-bit mask             */

#define  STS_CHK  0xe0  /*  status check value          */

#define  VAL_RET  0xd0  /*  value-retrieve control      */


unsigned  timer_rd (board, data_id)

int       board, data_id;

(
    int     cont_val;

    unsigned   value;

/*  ---------------------------------------------------*/
/*  Obtain the appropriate control value.              */
/*  ---------------------------------------------------*/
    switch (data_id)
        {
        case TMR_SAMP_PD:
            cont_val = CNT_0;
            break;
```

```
        case TMR_ONESHT:
            cont_val = CNT_1;
            break;

        case TMR_SAMP_CNT:
            cont_val = CNT_2;
            break;

        default:
            printf ("\nERROR!  timer_rd().\n");
            return (0);
            break;
        }

/*   ====================================================*/
/*   All i/o operations with the timer chip requires     */
/*   the setting of two address lines (A0, A1), which    */
/*   are settable from a control register.               */
/*   ----------------------------------------------------*/

/*   ----------------------------------------------------*/
/*   Write to control register the timer-control value   */
/*   (set A0 and A1).                                    */
/*   ----------------------------------------------------*/
     ctrl_wr (board, TMR_CNT, TMR_CTL_REG);


/*   ----------------------------------------------------*/
/*   Timer chip control register may now be read.        */
/*   Obtain the status byte . . .                        */
/*   ----------------------------------------------------*/
     bd_wr (board, TIMER, cont_val + STS_CHK, REGULAR);


/*   ----------------------------------------------------*/
/*   Manipulate A1 and A0 again to the appropriate       */
/*   counter.                                            */
/*   ----------------------------------------------------*/
     ctrl_wr (board, TMR_CNT, data_id);
     value = bd_rd (board, TIMER);
```

```
/*      ------------------------------------------------*/
/*      The status bit is active, thus indicating that  */
/*      the contents of the counter may be read.        */
/*      Write the appropriate control value to the      */
/*      control register (a0a1 = 11).                   */
/*      ------------------------------------------------*/

        ctrl_wr (board, TMR_CNT, TMR_CTL_REG);
        bd_wr (board, TIMER, cont_val + VAL_RET, REGULAR);

/*      Reset the address lines to the counter.         */
        ctrl_wr (board, TMR_CNT, data_id);

/*      Retrieve the LSB . . .                          */
        value = bd_rd (board, TIMER);

/*      . . .  then read and add the MSB.               */
        value += bd_rd (board, TIMER);
        }
    else
        {
/*      ------------------------------------------------*/
/*      The data could not be retrieved.  Since this    */
/*      function cannot indicate an error has occured,  */
/*      return the value of zero.                       */
/*      ------------------------------------------------*/
        value = 0;
        }

    return (value);
}
```

```
/*************************************************************
 *
 *  SOURCE:          timer_wr.c
 *
 *
 *  FUNCTION:        timer_wr (board, data_id, value)
 *
 *
 *  DESCRIPTION:     The purpose of this function is to
 *                   do a 16-bit write operation to the
 *                   timer/counter device present on the
 *                   A/D board (82C54-2).  The procedure
 *                   for programming this device is given
 *                   in the 1986 Intel Microprocessor Per-
 *                   ipheral Databook, p. 6-294.
 *
 *
 *  DOCUMENTATION
 *  FILES:           None.
 *
 *
 *  ARGUEMENTS:
 *
 *     board         (input) int
 *                   4-bit board address
 *
 *     data_id       (input) int
 *                   selects between the three counter
 *                   based on their function:
 *
 *                       TMR_SAMP_PD : sample period
 *                                     counter
 *                       TMR_ONESHT  : one-shot
 *                       TMR_SAMP_CNT: sample counter
 *
 *     value         (input) unsigned
 *                   the 16-bit number to be stored in the
 *                   specified counter
 *
 *
 *  RETURN:          (int)
 *
 *
```

```
 *  FUNCTIONS
 *  CALLED:         ctrl_wr (),
 *                  bd_wr (),
 *                  byte_brk ()
 *
 *
 *  AUTHOR:         Copyright 1989
 *                  Durwin D. Nigus
 *
 *
 *  DATE CREATED:   28 May 1989        Version 1.00
 *
 *  REVISIONS:      None.
 *
 *
**********************************************************/
#include <stdio.h>
#include <ctype.h>

#include "pi_ddn.h"

#define  SAMP_PERIOD_MODE   0x36   /* counter 0, mode 3 */

#define  ONESHOT_MODE       0x72   /* counter 1, mode 1 */
#define  SAMPLE_COUNT_MODE  0xb0   /* counter 2, mode 0 */

int   timer_wr (board, data_id, value)

int       board, data_id;

unsigned  value;

{
    int     cont_val, lsb, msb;


/*  -----------------------------------------------------*/
/*  Break  value  into its respective hi and low bytes. */
/*  -----------------------------------------------------*/
    byte_brk (value, &msb, &lsb);
```

```
/*  ---------------------------------------------------------*/
/*  Obtain the appropriate control value.                    */
/*  ---------------------------------------------------------*/
    switch (data_id)
        {
        case TMR_SAMP_PD:
            cont_val = SAMP_PERIOD_MODE;
            break;

        case TMR_ONESHT:
            cont_val = ONESHOT_MODE;
            break;

        case TMR_SAMP_CNT:
            cont_val = SAMPLE_COUNT_MODE;
            break;

        default:
            printf ("\nERROR!  timer_wr() \n");
            break;
        }

/*  =========================================================*/
/*  All i/o operations with the timer chip requires          */
/*  the setting of two address lines (A0, A1), which         */
/*  are settable from a control register.                    */
/*  ---------------------------------------------------------*/

/*  =========================================================*/
/*  Write to timer chip control register                     */
/*  the appropriate value (set A0, A1).                      */
/*  ---------------------------------------------------------*/
    ctrl_wr (board, TMR_CNT, TMR_CTL_REG);

/*  ---------------------------------------------------------*/
/*  Timer chip control register may now be written to.       */
/*  ---------------------------------------------------------*/
    bd_wr (board, TIMER, cont_val, REGULAR);
```

```
/*  ---------------------------------------------------*/
/*  Manipulate A1 and A0 again and write value to the  */
/*  appropriate counter--LSB first, then MSB.          */
/*  ---------------------------------------------------*/
    ctrl_wr (board, TMR_CNT, data_id);
    bd_wr (board, TIMER, lsb, REGULAR);
    bd_wr (board, TIMER, msb, REGULAR);

/*  ===================================================*/
/*  Both the LSB and MSB have now been written.        */
/*  ---------------------------------------------------*/
    return (NORMAL);
}
```

PCPI-DAS SOURCE CODE: ALTERED VERSIONS OF PCPI FUNCTIONS

```
/***********************************************************
 *
 *   SOURCE:          pi_cmd.c
 *
 *
 *   FUNCTION:        pi_cmd(buffer)
 *
 *
 *   DESCRIPTION:     Tokenizes the character string in
 *                    buffer ; observes the first three
 *                    tokens, treating them as follows:
 *                        First:  command
 *                        Second: register number (if req'd)
 *                        Third:  register value  (if req'd)
 *                    invokes the appropriate function
 *                    required to carry out the command.
 *
 *
 *
 *
 *   DOCUMENTATION
 *   FILES:           None.
 *
 *
 *   ARGUMENTS:
 *
 *       buffer       (input)  char *
 *                    Buffer containing the input
 *                    command string.
 *
 *   RETURN:          int
 *                    Command code.  -1 is returned if an
 *                    invalid command was entered.
 *
 *
 *   FUNCTIONS
 *   CALLED:
 *
 *
 *   AUTHOR:          Copyright 1989
 *                    Stephen A. Dyer, Ph.D.
 *
 *
 *   DATE CREATED:    19 March 1989       Version 1.00
 *
```

```
 *
 *  REVISIONS:      27Jul89  Removed 'C' comment command.
 *
 *
*********************************************************/
#include <stdio.h>
#include <string.h>

#include "pi_ddn.h"


#define MAX_TOKENS  3          /* Maximum number of tokens */
                               /* to extract.              */
#define LEN_TOKEN  10          /* Maximum length of each   */
                               /* token.                   */


int     pi_cmd(buffer)

char    *buffer;
{
    char    tokens[MAX_TOKENS][LEN_TOKEN + 1];
    char    *token;

    int     i, cmd_code, reg, token_no, value;

/*------------------------------------------------------*/
/*  Check to see if string is a comment.                */
/*------------------------------------------------------*/
    if (buffer[0]  == ';')
        {
/*      A comment.  Do not tokenize.                    */
        strcpy(pi_tempbuf, &buffer[1]);

        return (CMD_SEMI);
/*      Comment is in pi_tempbuf.                       */
        }


/*------------------------------------------------------*/
/*  Make a lower-case copy of  buffer .                 */
/*------------------------------------------------------*/
    strcpy(pi_tempbuf, buffer);
    strlwr(pi_tempbuf);
```

```
/*-------------------------------------------------------*/
/*  Tokenize the string.                                  */
/*-------------------------------------------------------*/
    for (token_no = 0; token_no < MAX_TOKENS; token_no++)
        {
        if (token_no == 0)
            token = strtok(pi_tempbuf, " ");
        else
            token = strtok(NULL, " ");

        if (token == NULL)
            break;

        strncpy(tokens[token_no], token, LEN_TOKEN);
        tokens[token_no][LEN_TOKEN] = '\0';
        }
/*      token_no  is now the number of tokens.            */


/*
    token_no = 0;
    token = strtok(pi_tempbuf, " ");
    strncpy(tokens[token_no], token, LEN_TOKEN);
    tokens[token_no][LEN_TOKEN] = '\0';

    while (token != NULL  &  token_no < MAX_TOKENS - 1)
        {
        token_no++;
        token = strtok(NULL, " ");
        strncpy(tokens[token_no], token, LEN_TOKEN);
        tokens[token_no][LEN_TOKEN] = '\0';
        }
*/

/*      token_no  is now the total number of tokens.      */


/*-------------------------------------------------------*/
/*  Save requested register and value, if pertinent.     */
/*-------------------------------------------------------*/
    if (token_no > 1)
        pi.fix[REG]   = pi_cvt(tokens[1]) % NUM_REGS;
```

```
    if (token_no > 2)
        pi.fix[VALUE] = pi_cvt(tokens[2]) % 256;


/*-------------------------------------------------------*/
/*  Determine command code.                               */
/*-------------------------------------------------------*/

    for (cmd_code = 1; cmd_code < NUM_CMDS; cmd_code++)
        {
        if (!strcmp(tokens[0],
                    commands[cmd_code].cmd_str))
            break;
        }

/*  If the entire command-table has been traversed        */
/*  without a match, then an error has occurred.          */
    if (cmd_code == NUM_CMDS)
        cmd_code = CMD_ERROR;


    return (cmd_code);
}
```

PCPI-DAS SOURCE CODE: ALTERED VERSIONS OF PCPI FUNCTIONS

```
/*****************************************************************
 *
 *   SOURCE:          pi_disp.c
 *
 *
 *   FUNCTION:        pi_disp(cmd_code, board)
 *
 *
 *   DESCRIPTION:     Command-dispatcher for pi test
 *                    commands.
 *                    Modified for the PCPI_DAS program.
 *                    Used with permission.
 *
 *
 *   DOCUMENTATION
 *   FILES:           None.
 *
 *
 *   ARGUMENTS:
 *
 *       cmd_code     (input)  int
 *                    Code for command to be dispatched.
 *
 *       board        (i/o) *int
 *                    Pointer to the board address value.
 *
 *
 *   RETURN:          (int)
 *                    cmd_code
 *
 *
 *   FUNCTIONS
 *   CALLED:
 *
 *
 *   AUTHOR:          Copyright 1989
 *                    Stephen A. Dyer, Ph.D.
 *
 *                    modified by :  Durwin D. Nigus
 *
 *
 *   DATE CREATED:    25 July 1989      Version 1.00
 *
 *
 *   REVISIONS:       None.
```

```
 *
 *
*************************************************************/

#include <stdio.h>
#include <stdlib.h>

#include "pi_ddn.h"
#include "p_plot.h"
#include "plot.h"


int     pi_disp(cmd_code, board)

int          cmd_code, *board;


{
    char        bin_buf[9], bin_buf2[9], prompt[PRMPT_MAX];

    int         first_list, i, reg, tmp_value, toggle;

    unsigned    conv_val, lsb, msb, value;
/*  ------------------------------------------------------*/
/*  Double-check  cmd_code  to assure that it is within */
/*  range.                                              */
/*  ------------------------------------------------------*/
    if (cmd_code < 0   ||   cmd_code >= NUM_CMDS)
        cmd_code = CMD_ERROR;



/*  ------------------------------------------------------*/
/*  No commands except 'enable', 'quit', comments, help */
/*  and 'si' are permitted when the i'face is disabled. */
/*  ------------------------------------------------------*/
    if (pi.fix[IFACE] == DISABLE)
        if (cmd_code == CMD_QUIT  ||
            cmd_code == CMD_SEMI  ||
            cmd_code == CMD_SI    ||
            cmd_code == CMD_E     ||
            cmd_code == CMD_HELP     )
        {
        }
    else
```

```
        {
        printf ("\n\n"
                "Interface is DISABLED! \n"
                "Command ignored. \n\n");
        cmd_code = CMD_ERROR;
        }


/*  ---------------------------------------------------------*/
/*  Find the matching routine.                               */
/*  ---------------------------------------------------------*/
    first_list = YES;

    switch (cmd_code)
        {
        case CMD_ERROR:
/*          ---------------------------------------------*/
/*          Error.                                       */
/*          ---------------------------------------------*/
            sprintf(pi_logbuf,
               commands[CMD_ERROR].log_format);
            break;


        case CMD_R:
/*          ---------------------------------------------*/
/*          Read register.                               */
/*          ---------------------------------------------*/
            pi.fix[VALUE] = pi_rdreg(pi.fix[REG]);

            ultoa((long)pi.fix[VALUE], bin_buf, 2);
            sprintf(pi_logbuf, commands[CMD_R].log_format,
                pi.fix[REG], pi.fix[VALUE], pi.fix[VALUE],
                bin_buf);
            break;


        case CMD_W:
/*          ---------------------------------------------*/
/*          Write register.                              */
/*          ---------------------------------------------*/
            pi_wrreg(pi.fix[REG], pi.fix[VALUE]);
```

```
                ultoa((long)pi.fix[VALUE], bin_buf, 2);
                sprintf(pi_logbuf, commands[CMD_W].log_format,
                    pi.fix[REG], pi.fix[VALUE], pi.fix[VALUE],
                    bin_buf);
                break;


            case CMD_SEMI:
/*              --------------------------------------------*/
/*              Comment.                                    */
/*              --------------------------------------------*/
                sprintf(pi_logbuf,
                   commands[CMD_SEMI].log_format, pi_tempbuf);
                break;


            case CMD_E:
/*              --------------------------------------------*/
/*              Enable interface.                           */
/*              --------------------------------------------*/
                pi_iface(ENABLE);
                sprintf(pi_logbuf,
                   commands[CMD_E].log_format);
                break;


            case CMD_D:
/*              --------------------------------------------*/
/*              Disable interface.                          */
/*              --------------------------------------------*/
                pi_iface(DISABLE);
                sprintf(pi_logbuf,
                   commands[CMD_D].log_format);
                break;


            case CMD_I:
```

```
/*          ------------------------------------------------*/
/*          Toggle initialization.                          */
/*          ------------------------------------------------*/
            if (pi.fix[MRESET] == ENABLE  ||
                pi.fix[BDINIT] == ENABLE)
                {
                toggle = OFF;
                pi_initl(DISABLE, DISABLE);
                }
            else
                {
                toggle = ON;
                pi_initl(ENABLE, ENABLE);
                }
            sprintf(pi_logbuf, commands[CMD_I].log_format,
                pi_prsta("INACTIVE", "ACTIVE", toggle));
            break;


        case CMD_IB:
/*          ------------------------------------------------*/
/*          Activate -RFINTCB.                              */
/*          ------------------------------------------------*/
            pi_initl(DISABLE, ENABLE);
            sprintf(pi_logbuf,
               commands[CMD_IB].log_format);
            break;


        case CMD_IM:
/*          ------------------------------------------------*/
/*          Activate -MINIT.                                */
/*          ------------------------------------------------*/
            pi_initl(ENABLE, DISABLE);
            sprintf(pi_logbuf,
               commands[CMD_IM].log_format);
            break;
```

```
        case CMD_IA:
/*          -------------------------------------------*/
/*          Activate all inits.                        */
/*          -------------------------------------------*/
            pi_initl(ENABLE, ENABLE);
            sprintf(pi_logbuf,
               commands[CMD_IA].log_format);
            break;


        case CMD_ID:
/*          -------------------------------------------*/
/*          Deactivate all inits.                      */
/*          -------------------------------------------*/
            pi_initl(DISABLE, DISABLE);
            sprintf(pi_logbuf,
               commands[CMD_ID].log_format);
            break;


        case CMD_L:
/*          -------------------------------------------*/
/*          Toggle session-logger.                     */
/*          -------------------------------------------*/
            if (pi.fix[LOG] == OFF)
                pi.fix[LOG] = ON;
            else
                pi.fix[LOG] = OFF;

            sprintf(pi_logbuf, commands[CMD_L].log_format,
                pi_prsta("OFF", "ON", pi.fix[LOG]));
            break;


        case CMD_QUIT:
/*          -------------------------------------------*/
/*          Terminate session.                         */
/*          -------------------------------------------*/
            pi.fix[QUIT] = YES;
            pi_iface(DISABLE);
            pi_date();
            sprintf(pi_logbuf,
```

```
                  commands[CMD_QUIT].log_format,
                     pi.log_date);
               break;


          case CMD_P:
/*             ---------------------------------------------*/
/*             Plot contents of data buffer on display.     */
/*             ---------------------------------------------*/
               if (pre_plot () != ERR)
                   int_plot(DISPLAY, plot_len, 0, 1,
                            pi_data, "Component Number", "",
                            "Amplitude", "",
                            "PCPI Data Buffer", CURVE, 0.0);

                   sprintf(pi_logbuf,
                       commands[CMD_P].log_format);
               break;


          case CMD_PP:
/*             ---------------------------------------------*/
/*             Plot contents of data buffer on display.     */
/*             ---------------------------------------------*/
               if (pre_plot () != ERR)
                   {
                   printf ("Enter plot title:  ");
                   pi_gets(50, pi_buf1);
                   int_plot(PLOTTER, plot_len, 0, 1,
                            pi_data, "Component Number", "",
                            "Amplitude", "", pi_buf1, CURVE,
                            10.0);
                   }

               sprintf(pi_logbuf,
                  commands[CMD_PP].log_format, pi_buf1);
               break;
```

```
/*==========================================================*/
/*      ddn modifications follow                            */
/*----------------------------------------------------------*/
        case CMD_HELP:
/*          ----------------------------------------------*/
/*          Show the list of commands and their           */
/*          descriptions.                                 */
/*          ----------------------------------------------*/
            help_das ();
            break;


        case CMD_TR:
/*          ----------------------------------------------*/
/*          Trace-mode toggler.                           */
/*          ----------------------------------------------*/
            printf ("Trace mode toggle. \n\n");

            trace = (trace == ON) ? (OFF) : (ON);

            if (trace == OFF)
                printf ("Trace mode is now OFF. \n\n");
            else
                printf ("Trace mode is now ON. \n\n");

            sprintf(pi_logbuf,
               commands[CMD_TR].log_format);

            break;


        case CMD_STEP:
/*          ----------------------------------------------*/
/*          Trace-step toggler.                           */
/*          ----------------------------------------------*/
            if (trace == OFF)
                printf ("Trace must be enabled for "
                        "stepper toggle. \n\n");
            else
                {
                printf ("Stepper mode toggle. \n\n");

                stepper = (stepper == ON) ? (OFF) : (ON);
```

```
                if (stepper == OFF)
                    printf ("Stepper mode is now OFF."
                            "\n\n");
                else
                    printf ("Stepper mode is now ON."
                        "\n\n"
                        "When an I/O statement appears, "
                        "press a character followed by "
                        "<enter> \n\n");

                }
            sprintf(pi_logbuf,
               commands[CMD_STEP].log_format);

            break;


        case CMD_SI:
/*          -------------------------------------------*/
/*          Initalize system                           */
/*          -------------------------------------------*/

/*          Enable PCPI circuit.                       */

            pi_iface(ENABLE);
            sprintf(pi_logbuf,
                    commands[CMD_E].log_format);

/*          initialize bus drivers                     */
            si_das ();

/*          Tell which boards are present.             */
/*          If no boards are present, cease initialize. */

            if (bp_das () != 0)
                {
/*              select board address                   */

                *board = 1;
                printf("Enter desired board address (%d):"
                       "  ", *board);
                if (pi_gets(BUF_LEN-1, pi_buf1) == 0)
                    tmp_value = *board;
                else
```

```
                    tmp_value = pi_cvt(pi_buf1);

                tmp_value = (tmp_value > 0) ?
                             tmp_value : 0;
                *board =    (tmp_value < 16) ?
                            tmp_value : 0;

/*              Initalize selected board.                    */
                bi_das (*board);
                )

            sprintf(pi_logbuf,
               commands[CMD_SI].log_format);

            break;


        case CMD_BI:
/*          ------------------------------------------------*/
/*          Initialize board                                 */
/*          ------------------------------------------------*/
            bi_das (*board);
            sprintf(pi_logbuf,
               commands[CMD_BI].log_format);
            break;


        case CMD_CT:
/*          ------------------------------------------------*/
/*          Configure system trigger                         */
/*          ------------------------------------------------*/
            ct_das ();
            sprintf(pi_logbuf,
               commands[CMD_CT].log_format);
            break;


        case CMD_CC:
/*          ------------------------------------------------*/
/*          Configure system clock                           */
/*          ------------------------------------------------*/
            cc_das ();
```

```
            sprintf(pi_logbuf,
               commands[CMD_CC].log_format);
            break;


        case CMD_BP:
/*          ---------------------------------------------*/
/*          Determine boards present on bus.             */
/*          ---------------------------------------------*/
            bp_das ();
            sprintf(pi_logbuf,
               commands[CMD_BP].log_format);
            break;


        case CMD_DR:
/*          ---------------------------------------------*/
/*          Display bus driver registers.                */
/*          ---------------------------------------------*/
            dr_das ();
            sprintf(pi_logbuf,
               commands[CMD_DR].log_format);
            break;


        case CMD_SB:
/*          ---------------------------------------------*/
/*          Set board address value.                     */
/*          ---------------------------------------------*/
            tmp_value = pi.fix[REG];

            tmp_value = (tmp_value > 0) ? tmp_value : 0;
            *board =  (tmp_value < 16) ?  tmp_value : 0;

            sprintf(pi_logbuf,
               commands[CMD_SB].log_format);
            break;


        case CMD_BR:
/*          ---------------------------------------------*/
/*          Read register from board.                    */
/*          ---------------------------------------------*/
            reg = (pi.fix[REG] & 0x0f) + DATA_BOTH;
            value = bd_rd (*board, reg);
```

```
            reg &= 0x0f;
            byte_brk (value, &msb, &lsb);
            ultoa ((long)msb, bin_buf, 2);
            ultoa ((long)lsb, bin_buf2, 2);

            printf ("\n\n"
                "Board %i, reg %i contents = %4x h, "
                "%5u dec, %8s %8s b\n\n", *board,
                reg, value, value, bin_buf, bin_buf2);
            sprintf(pi_logbuf,
               commands[CMD_BR].log_format);
            break;


        case CMD_BW:
/*          ---------------------------------------------*/
/*          Write to register on board.                  */
/*          ---------------------------------------------*/
            reg = (pi.fix[REG] & 0x0f) + DATA_BOTH;
            value = pi.fix[VALUE];
            bd_wr (*board, reg, value, REGULAR);

/*          Echo value written by reading it.            */
            value = bd_rd (*board, reg);
            reg &= 0x0f;
            byte_brk (value, &msb, &lsb);
            ultoa ((long)msb, bin_buf, 2);
            ultoa ((long)lsb, bin_buf2, 2);

            printf ("\n\n"
                "Board %i, reg %i contents = %x h, "
                "%u dec, %s %s b\n\n", *board,
                reg, value, value, bin_buf, bin_buf2);
            sprintf(pi_logbuf,
               commands[CMD_BW].log_format);
            break;


        default:
/*      -------------------------------------------------*/
/*      Set flag that indicates that the instruction    */
/*      was not found in this list.                      */
/*      -------------------------------------------------*/
            first_list = NO;
            break;
```

```
        }

/*  ------------------------------------------------------*/
/*  The following routines are for a specific board and */
/*  thus the board needs to be initialized before it    */
/*  can have bus i/o.                                    */
/*  ------------------------------------------------------*/
    if (initialized[*board] == NO && first_list == NO)
        {
        printf ("The board is not initialized.  "
                "Use 'bi' instruction. \n");

        sprintf(pi_logbuf,
           commands[CMD_ERROR].log_format);
        return (CMD_ERROR);
        }

/*  ------------------------------------------------------*/
/*  Now examine the board-specific commands.             */
/*  ------------------------------------------------------*/
    switch (cmd_code)
        {
        case CMD_TS:
/*          ------------------------------------------------*/
/*          Select trigger for specified board.            */
/*          ------------------------------------------------*/
            ts_das (*board);
            sprintf(pi_logbuf,
               commands[CMD_TS].log_format);
            break;

        case CMD_CS:
/*          ------------------------------------------------*/
/*          Select clock for specified board.              */
/*          ------------------------------------------------*/
            cs_das (*board);
            sprintf(pi_logbuf,
               commands[CMD_CS].log_format);
            break;


        case CMD_FS:
```

```
/*          ------------------------------------------*/
/*          Set full-scale signal range.              */
/*          ------------------------------------------*/
            fs_das (*board);
            sprintf(pi_logbuf,
               commands[CMD_FS].log_format);
            break;


        case CMD_FI:
/*          ------------------------------------------*/
/*          Enable or disable the on-board filter.    */
/*          ------------------------------------------*/
            fi_das (*board);
            sprintf(pi_logbuf,
               commands[CMD_FI].log_format);
            break;


        case CMD_SR:
/*          ------------------------------------------*/
/*          Set sampling rate for specified board.    */
/*          ------------------------------------------*/
            sr_das (*board);
            sprintf(pi_logbuf,
               commands[CMD_SR].log_format);
            break;


        case CMD_BC:
/*          ------------------------------------------*/
/*          Begin conversion on specified board.      */
/*          ------------------------------------------*/
            bc_das (*board);
            sprintf(pi_logbuf,
               commands[CMD_BC].log_format);
            break;


        case CMD_SC:
/*          ------------------------------------------*/
/*          Stop conversion immediately on            */
/*          specified board.                          */
/*          ------------------------------------------*/
            sc_das (*board);
```

```
            sprintf(pi_logbuf,
               commands[CMD_SC].log_format);
            break;


        case CMD_GS:
/*          -------------------------------------------*/
/*          Display status for specified board.         */
/*          -------------------------------------------*/
            gs_das (*board);
            sprintf(pi_logbuf,
               commands[CMD_GS].log_format);
            break;


        case CMD_RS:
/*          -------------------------------------------*/
/*          Retrieve samples from specified board.      */
/*          -------------------------------------------*/
            rs_das (*board);
            sprintf(pi_logbuf,
               commands[CMD_RS].log_format);
            break;


        case CMD_MT:
/*          -------------------------------------------*/
/*          Perform on-board memory test.               */
/*          -------------------------------------------*/
            mt_das (*board);
            sprintf(pi_logbuf,
               commands[CMD_MT].log_format);
            break;


        case CMD_GV:
/*          -------------------------------------------*/
/*          Get A/D conversion value with a specified   */
/*          source.                                     */
/*          -------------------------------------------*/
            conv_val = (unsigned)gv_das (*board);
            byte_brk (conv_val, &msb, &lsb);
```

```
            ultoa((long)msb, bin_buf, 2);
            ultoa((long)lsb, bin_buf2, 2);
            printf ("\n"
                    "The conversion value is: \n\n"
                    "%6d (signed) %5u (unsigned)  %4x"
                    " (hex)  %8s %8s (bin)\n\n", conv_val,
                    conv_val, conv_val, conv_val,
                    bin_buf, bin_buf2);
            sprintf(pi_logbuf,
                     commands[CMD_GV].log_format);
            break;


        case CMD_CL:
/*          ---------------------------------------------*/
/*          Perform calibration routine.                 */
/*          ---------------------------------------------*/
            cl_das (*board);
            sprintf(pi_logbuf,
               commands[CMD_CL].log_format);
            break;


        case CMD_FC:
/*          ---------------------------------------------*/
/*          Perform filter configuration.                */
/*          ---------------------------------------------*/
            fc_das (*board);
            sprintf(pi_logbuf,
               commands[CMD_FC].log_format);
            break;

        }

    return (cmd_code);
}
```

```
/************************************************************
 *
 *  SOURCE:          pi_init.c
 *
 *
 *  FUNCTION:        pi_init();
 *
 *
 *  DESCRIPTION:     This function performs the
 *                   configuration and initialization
 *                   necessary before any functions which
 *                   access external module are invoked.
 *                   Modified for the PCPI_DAS program.
 *
 *
 *  DOCUMENTATION
 *  FILES:           None.
 *
 *
 *  ARGUMENTS:       None.
 *
 *
 *  RETURN:          int
 *                   0:          Normal return.
 *                   ERR_INIT    An error occurred.
 *
 *
 *  FUNCTIONS
 *  CALLED:
 *
 *
 *  AUTHOR:          Copyright 1989
 *                   Stephen A. Dyer, Ph.D.
 *
 *                   Modified for use with the DAS
 *                   by Durwin D. Nigus.
 *
 *
 *
 *  DATE CREATED:    19 March 1989        Version 1.00
 *
 *
 *  REVISIONS:       20Mar89 Add strcpy to [CMD_AD].
 *                    5Apr89 Add default filename for
 *                           output-data file.
```

```
 *                                Initialize lengths of data
 *                                buffer and files.  Set up
 *                                conversion tables for sampling
 *                                rates.
 *                                Zero data buffer.
 *                                Add default sampling rates.
 *                                Add CMD_B to list.
 *                                Add CMD_P to list.
 *                         9Apr89 Add CMD_P to list.
 *                        10Apr89 Change command "ir" to "ib".
 *                                Change log string for CMD_P.
 *
 *                         4Jul89 Modified for PCPI_DAS.
 *
*********************************************************/

#define PI_INIT     1

#include <stdio.h>
#include <string.h>
#include <stdlib.h>

#include "pi_ddn.h"


int     pi_init()
{
    char    *getenv(const char *),
            *pi_baseport,
            *test_it,
            *xtl,
            *dummy;

    int     i;


/*  --------------------------------------------------------*/
/*  Set base_port.                                          */
/*  --------------------------------------------------------*/
    if ((pi_baseport = getenv("PCPI_BASE")) == NULL)
        return (ERR_INITBP);

    pi.base_port = (unsigned)pi_cvt(pi_baseport);
```

```
/*  --------------------------------------------------------*/
/*  Set test_das.                                           */
/*  --------------------------------------------------------*/
    if ((test_it = getenv("TEST_DAS")) == NULL)
        return (ERR_INITTST);

    test_das = (pi_cvt(test_it) == 0) ? (NO) : (YES);


/*  --------------------------------------------------------*/
/*  Set A/D board internal oscillator frequency.            */
/*  --------------------------------------------------------*/
    if ((xtl = getenv("XTAL_FREQ")) == NULL)
        return (ERR_INITXTL);

    xtal_freq = strtol (xtl, &dummy, 10);


/*  --------------------------------------------------------*/
/*  Disable interface.                                      */
/*  --------------------------------------------------------*/
    pi_iface(DISABLE);


/*  --------------------------------------------------------*/
/*  Deactivate -MRESET and -BDINIT.                         */
/*  --------------------------------------------------------*/
    pi_initl(INACTIVE, INACTIVE);


/*  --------------------------------------------------------*/
/*  Activate and initialize screen-logger and session-      */
/*  logger.                                                 */
/*  --------------------------------------------------------*/
    pi.fix[SCREEN]   = ON;
    pi.fix[LOG]      = ON;

    pi.fix[LOG_NO]   = 0;
    pi.fix[LAST_LOG] = pi.fix[LOG_NO];
```

```
/*  ------------------------------------------------------*/
/*  Put a default filename for the session-logger into   */
/*   pi.log_fn .                                          */
/*  ------------------------------------------------------*/
    strcpy(pi.log_fn, "pcpi.log");


/*  ------------------------------------------------------*/
/*  Put a default filename for the output-data file      */
/*   pi.out_file .                                        */
/*  ------------------------------------------------------*/
    strcpy(pi.out_file, "pcpi.out");


/*  ------------------------------------------------------*/
/*  Initialize  buffer and file lengths.                  */
/*  ------------------------------------------------------*/
/*  Default length of data buffer.                        */
    pi.fix[DATA_LEN] = 0;

/*  Length of input-data file.                            */
    pi.fix[LEN_INFILE] = 0;

/*  Length of plot data buffer.                           */
    plot_len = 0;

/*  ------------------------------------------------------*/
/*  Zero contents of data buffer.                         */
/*  ------------------------------------------------------*/
    for (i = 0; i < MAX_DATA; i++)
        pi_data[i] = 0;


/*  ------------------------------------------------------*/
/*  Set board initialization state to NO.                 */
/*  ------------------------------------------------------*/
    for (i = 0; i < MAX_BDS; i++)
        initialized[i] = NO;


/*  ------------------------------------------------------*/
/*  Turn off tracer and stepper.                          */
/*  ------------------------------------------------------*/
    trace =   NO;
    stepper = NO;
```

```
/*  ---------------------------------------------------*/
/*  Zero the sample counters.                           */
/*  ---------------------------------------------------*/
    num_pre_trig  = 0;
    num_post_trig = 0;


/*  ---------------------------------------------------*/
/*  Initialize  commands[] .                            */
/*  ---------------------------------------------------*/
    strcpy(commands[CMD_ERROR].cmd_str, "");
    strcpy(commands[CMD_ERROR].log_format,
        "Error:  invalid command entered.");

    strcpy(commands[CMD_R].cmd_str, "r");
    strcpy(commands[CMD_R].log_format,
        "Read from register %2d the value %3d = %2.2Xh = "
        "%8.8sb.");

    strcpy(commands[CMD_W].cmd_str, "w");
    strcpy(commands[CMD_W].log_format,
        "Write to register  %2d the value %3d = %2.2Xh = "
        "%8.8sb.");

    strcpy(commands[CMD_SEMI].cmd_str, ";");
    strcpy(commands[CMD_SEMI].log_format,
        "COMMENT:  %.40s");

    strcpy(commands[CMD_E].cmd_str, "e");
    strcpy(commands[CMD_E].log_format,
        "Enable interface.");

    strcpy(commands[CMD_D].cmd_str, "d");
    strcpy(commands[CMD_D].log_format,
        "Disable interface.");

    strcpy(commands[CMD_I].cmd_str, "i");
    strcpy(commands[CMD_I].log_format,
        "Toggle initialization %s.");

    strcpy(commands[CMD_IB].cmd_str, "ib");
    strcpy(commands[CMD_IB].log_format,
        "Activate -BDINIT.");
```

```
    strcpy(commands[CMD_IM].cmd_str, "im");
    strcpy(commands[CMD_IM].log_format,
        "Activate -MRESET.");

    strcpy(commands[CMD_IA].cmd_str, "ia");
    strcpy(commands[CMD_IA].log_format,
        "Activate -MRESET and -BDINIT.");

    strcpy(commands[CMD_ID].cmd_str, "id");
    strcpy(commands[CMD_ID].log_format,
        "Deactivate -MRESET and -BDINIT.");

    strcpy(commands[CMD_L].cmd_str, "l");
    strcpy(commands[CMD_L].log_format,
        "Toggle session logger %s.");

    strcpy(commands[CMD_QUIT].cmd_str, "quit");
    strcpy(commands[CMD_QUIT].log_format,
        "PCPI session terminated %s.");

    strcpy(commands[CMD_P].cmd_str, "p");
    strcpy(commands[CMD_P].log_format,
        "Plot contents of data buffer on display.");

    strcpy(commands[CMD_PP].cmd_str, "pp");
    strcpy(commands[CMD_PP].log_format,
        "Plot contents of data buffer on pen plotter.  \n"
        "       Title:  %s");

/*  ====================================================*/
/*  ddn commands                                        */
/*  ----------------------------------------------------*/
    strcpy(commands[CMD_SI].cmd_str, "si");
    strcpy(commands[CMD_SI].log_format, "Initialize "
                                         "system.");

    strcpy(commands[CMD_BI].cmd_str, "bi");
    strcpy(commands[CMD_BI].log_format, "Initialize "
                                         "board. ");

    strcpy(commands[CMD_CT].cmd_str, "ct");
    strcpy(commands[CMD_CT].log_format, "Configure "
                                         "system trigger");
```

```
strcpy(commands[CMD_CC].cmd_str, "cc");
strcpy(commands[CMD_CC].log_format, "Configure "
                                     "system clock.");

strcpy(commands[CMD_TS].cmd_str, "ts");
strcpy(commands[CMD_TS].log_format, "A/D board "
                                   "trigger select.");

strcpy(commands[CMD_CS].cmd_str, "cs");
strcpy(commands[CMD_CS].log_format, "A/D board clock "
                                     "select.");

strcpy(commands[CMD_FS].cmd_str, "fs");
strcpy(commands[CMD_FS].log_format, "A/D board gain "
                                     "set.");

strcpy(commands[CMD_FI].cmd_str, "fi");
strcpy(commands[CMD_FI].log_format, "A/D board "
                              "filter control.");

strcpy(commands[CMD_SR].cmd_str, "sr");
strcpy(commands[CMD_SR].log_format, "A/D board "
                               "sample rate control.");

strcpy(commands[CMD_BC].cmd_str, "bc");
strcpy(commands[CMD_BC].log_format, "A/D board: "
                           "begin conversion.");

strcpy(commands[CMD_SC].cmd_str, "sc");
strcpy(commands[CMD_SC].log_format, "A/D board: "
                           "stop conversion.");

strcpy(commands[CMD_GS].cmd_str, "gs");
strcpy(commands[CMD_GS].log_format, "A/D board:  "
                                     "get status.");

strcpy(commands[CMD_BP].cmd_str, "bp");
strcpy(commands[CMD_BP].log_format, "Query for "
                              "boards present.");

strcpy(commands[CMD_DR].cmd_str, "dr");
strcpy(commands[CMD_DR].log_format, "Display bus "
                                     "driver ports.");
```

```
    strcpy(commands[CMD_SB].cmd_str, "sb");
    strcpy(commands[CMD_SB].log_format, "Set board "
                                        "value.");

    strcpy(commands[CMD_TR].cmd_str, "tr");
    strcpy(commands[CMD_TR].log_format, "Toggle trace "
                                           "mode.");

    strcpy(commands[CMD_STEP].cmd_str, "step");
    strcpy(commands[CMD_STEP].log_format, "Step mode.");

    strcpy(commands[CMD_RS].cmd_str, "rs");
    strcpy(commands[CMD_RS].log_format, "A/D board: "
                                        "retrieve "
                                        "samples. ");

    strcpy(commands[CMD_MT].cmd_str, "mt");
    strcpy(commands[CMD_MT].log_format, "A/D board:  "
                                 "memory test.");

    strcpy(commands[CMD_GV].cmd_str, "gv");
    strcpy(commands[CMD_GV].log_format, "A/D board: get "
                                        "value.");

    strcpy(commands[CMD_CL].cmd_str, "cl");
    strcpy(commands[CMD_CL].log_format, "A/D board: "
                                        "calibrate.");

    strcpy(commands[CMD_FC].cmd_str, "fc");
    strcpy(commands[CMD_FC].log_format, "Filter coeff. "
                                        "retrieve.");

    strcpy(commands[CMD_BR].cmd_str, "br");
    strcpy(commands[CMD_BR].log_format, "Board read.");

    strcpy(commands[CMD_BW].cmd_str, "bw");
    strcpy(commands[CMD_BW].log_format, "Board write.");

    strcpy(commands[CMD_HELP].cmd_str, "help");
    strcpy(commands[CMD_HELP].log_format,
                                 "Show help-key list.");

    return (0);
}
```

DESIGN OF AN
EASY-TO-USE, HOST-INDEPENDENT
DATA ACQUISITION SYSTEM

by

DURWIN DUANE NIGUS

B.S., Kansas State University, 1987

AN ABSTRACT OF A THESIS

submitted in partial fulfillment of the

requirements for the degree

MASTER OF SCIENCE

ELECTRICAL AND COMPUTER ENGINEERING

KANSAS STATE UNIVERSITY
Manhattan, Kansas

1989

ABSTRACT

A design is presented for a host-independent data acquisition system. This acquisition system has the advantage of being usable by virtually any computer that has an RS-232 port. The motivation for the system's development was to reduce the cumbersome interface problems frequently encountered when moving an acquisition system from one host computer to another.

The thesis describes a system that is modular in the sense that it is composed of a system controller and up to sixteen removable I/O boards, all of which are interconnected by a system bus. The system controller receives mnemonic commands from the host computer by way of an RS-232 communication link, whereupon the command is deciphered and the appropriate actions are taken. An important goal during the design of this system was to minimize complexity of the host computer's system-controlling software, thus reducing the development time when the system is used with a new host.

The thesis presents the design specifications for the system, the design of the system, a design for an analog-to-digital (A/D) board, and algorithms used to control the system. Additionally, a list of system commands, an A/D-board user's guide, and software used for system testing are included as appendices. This thesis also contains

sufficient information to facilitate the design and construction of an I/O board compatible with the system.

The A/D board designed for use in the system has many features. The board has a differential signal input with programmable gain. A programmable filter is provided on-board for anti-aliasing and other signal conditioning needs. A/D conversions are made by a 12-bit, bipolar successive-approximation ADC. The sampling rate is programmable from 0.2 Hz to 150 kHz. On-board memory retains up to 64K samples during an acquisition sequence. In addition, the A/D board has flexible trigger selection as well as pre-trigger sampling capabilities.