ACCES I/O PRODUCTS INC
10623 Roselle Street, San Diego, CA 92121
TEL (858)550-9559 FAX (858)550-7322

MODEL AIO8

USER MANUAL

FILE: MAIO8.B5

Notice

The information in this document is provided for reference only. ACCES does not assume any liability arising out of the application or use of the information or products described herein. This document may contain or reference information and products protected by copyrights or patents and does not convey any license under the patent rights of ACCES, nor the rights of others.

IBM PC, PC/XT, and PC/AT are registered trademarks of the International Business Machines Corporation.

Printed in USA. Copyright 1995 by ACCES I/O Products Inc, 10623 Roselle Street, San Diego, CA 92121. All rights reserved.

Warranty

Prior to shipment, ACCES equipment is thoroughly inspected and tested to applicable specifications. However, should equipment failure occur, ACCES assures its customers that prompt service and support will be available. All equipment originally manufactured by ACCES which is found to be defective will be repaired or replaced subject to the following considerations.

Terms and Conditions

If a unit is suspected of failure, contact ACCES' Customer Service department. Be prepared to give the unit model number, serial number, and a description of the failure symptom(s). We may suggest some simple tests to confirm the failure. We will assign a Return Material Authorization (RMA) number which must appear on the outer label of the return package. All units/components should be properly packed for handling and returned with freight prepaid to the ACCES designated Service Center, and will be returned to the customer's/user's site freight prepaid and invoiced.

Coverage

First Three Years: Returned unit/part will be repaired and/or replaced at ACCES option with no charge for labor or parts not excluded by warranty. Warranty commences with equipment shipment.

Following Years: Throughout your equipment's lifetime, ACCES stands ready to provide on-site or in-plant service at reasonable rates similar to those of other manufacturers in the industry.

Equipment Not Manufactured by ACCES

Equipment provided but not manufactured by ACCES is warranted and will be repaired according to the terms and conditions of the respective equipment manufacturer's warranty.

General

Under this Warranty, liability of ACCES is limited to replacing, repairing or issuing credit (at ACCES discretion) for any products which are proved to be defective during the warranty period. In no case is ACCES liable for consequential or special damage arriving from use or misuse of our product. The customer is responsible for all charges caused by modifications or additions to ACCES equipment not approved in writing by ACCES or, if in ACCES opinion the equipment has been subjected to abnormal use. "Abnormal use" for purposes of this warranty is defined as any use to which the equipment is exposed other than that use specified or intended as evidenced by purchase or sales representation. Other than the above, no other warranty, expressed or implied, shall apply to any and all such equipment furnished or sold by ACCES.

Table of Contents

Notice . . . iii
Warranty . . . ii

Chapter 1: Introduction . . . 1-1
- Analog Inputs . . . 1-1
- Input System Expansion . . . 1-1
- Reference Voltage Output . . . 1-1
- Counter/Timer . . . 1-2
- Interrupts . . . 1-2
- Utility Software . . . 1-2
- Enhancements . . . 1-3
- Specifications . . . 1-5

Chapter 2: Installation . . . 2-1
- CD Installation . . . 2-1
- 3.5-Inch Diskette Installation . . . 2-1
- Directories Created on the Hard Disk . . . 2-2
- Configuration File . . . 2-4

Chapter 3: Hardware Configuration and Installation . . . 3-1
- Option Selection . . . 3-1
- Interrupts . . . 3-1
- Selecting a Base Address . . . 3-1
- Setting the Base Address . . . 3-4
- Using the Setup Program to Set the Base Address . . . 3-4
- Installing the Card . . . 3-5
- Calibration and Test . . . 3-6

Chapter 4: Programming . . . 4-1
- AIO8 Register Address Map . . . 4-1
- Register Definitions . . . 4-2
- Programming Using the Driver . . . 4-5

Chapter 5: AIO8 Driver Reference . . . 5-1
- Using the Driver . . . 5-1
- Task Summary . . . 5-2
- Task 0: Initialize . . . 5-3
- Task 1: Check A/D Operations . . . 5-4
- Task 2: Fetch Gain Code for a Point Address . . . 5-4
- Task 3: Fetch Point Address for a Point List Index . . . 5-5
- Task 4: Assign Gain Code to Range of Point Addresses . . . 5-6
- Task 5: Assign Point Addresses to the Point List . . . 5-7
- Task 6: Fetch Data from a Point Address . . . 5-8
- Task 7: Fetch Data from next Point Address in List . . . 5-8

Task 8: Fetch Multiple Buffered Conversions 5-9
Task 9: Interrupt Driven Data Acquisition 5-10
Task 10: Thermocouple/Function Assignment 5-12
Task 11: Reset Functions 5-15
Task 12: Digital Output 5-16
Task 13: Digital Input 5-16
Task 14: Counter/Timer Setup 5-17
Task 15: Read Counter/Timer Count 5-18
Task 16: Measure Frequency 5-18
Task 17: Measure Pulse Width 5-20
Summary of Error Codes 5-21

Chapter 6: A/D Converter Applications 6-1
Connecting Analog Inputs 6-1
Comments on Noise Interference 6-1
Input Range and Resolution Specifications 6-2
Current Measurements 6-2
Measuring Large Voltages 6-2
Adding More Analog Inputs 6-3
Precautions - Noise, Ground Loops, and Overloads 6-3

Chapter 7: Programmable Interval Timer 7-1
Operational Modes 7-1
Programming 7-2
Reading and Loading the Counters 7-3
Programming Examples 7-4

Appendix A: Linearization A-1

Appendix B: Cabling and Connector Information B-1
AIO8 Output Connector Pin Assignments B-1

Appendix C: Basic Integer Variable Storage C-1

List of Figures

Figure 1-1: AIO8 Block Diagram 1-7
Figure 3-1: Option Selection Map 3-2

List of Tables

Table 2-1: Configuration File Example 2-5
Table 3-1: Standard Address Assignments for 286/386/486 Computers 3-3
Table 3-2: Base Address Example 3-4
Table 4-1: AIO8 Register Address Map 4-1
Table B-1: AIO8 Output Connector Pin Assignments B-1

Chapter 1: Introduction

The AIO8 is a multifunction, moderate-speed analog-to-digital converter card with counter/timers. This card may be used in IBM Personal Computers and other compatible computers. The card requires one slot in the computer. All external connections are made through a standard 37-pin D-type connector at the rear of the computer. The following paragraphs describe the functions provided by the AIO8 card.

Analog Inputs

The card accepts eight single-ended analog input channels. The full scale input for all channels is ±5V (0.00244V resolution). Inputs are single-ended with a common ground and can withstand over voltages up to ±30 volts and brief transients of several hundred volts. When power is off, the inputs are open-circuited providing fail-safe operation. The analog-to-digital converter (A/D) is a 12-bit successive-approximation type with a sample and hold input. Conversion time is typically 25µS (35µS maximum) and, depending on the speed of the software and computer platform, throughputs of up to 4,000 conversions per second are attainable.

Input System Expansion

The AIO8 card may be used alone or it can support up to eight AIM-16 or LVDT-8 analog input expansion cards. An 4-bit standard LSTTL logic output from the AIO8 is used to select one of 16 analog input channels at the AIM-16. When interfacing to the LVDT-8, three bits are used to select one of eight LVDT-8 inputs. Since the eight-input multiplexer on the AIO8 card is software addressable, an input expansion card may be connected to each input, for a maximum of 128 channels in the system. If more than 128 analog inputs are required, a second AIO8 with companion input expansion cards can be used.

Reference Voltage Output

A precision +10.0V (±0.1V) reference voltage output is derived from the A/D converter reference and is available as an output from the AIO8. The reference voltage can source or sink up to 2mA.

PC bus power (+5V, +12V, -12V) is also provided at the rear connector. This allows additional user-designed interfaces for input signal conditioning, expansion multiplexers, etc.

Counter/Timer

The AIO8 includes a type 8254 counter/timer which has three 16-bit programmable down counters. This chip is used for event counting, pulse and waveform generation, frequency and period measurement, etc. A/D conversion cycles be initiated by the Counter/Timer when a jumper is installed on the I/O connector. Software provided includes the means to use these counters to provide programmable gain commands to the instrumentation amplifier on the AIM-16 expansion sub-multiplexer card. Refer to Programmable Interval Timer for a description of the functions of the 8254 counter/timer chip.

Interrupts

Interrupts are supported from either external inputs or from the end-of-conversion signal from the A/D converter. Selection of the interrupt levels (IRQ2-7), is made by jumper. Interrupts are software enabled and disabled. An interrupt request may be canceled by either of the following signals:

a. The computer reset signal.
b. The writing of a command word to the card. That is, either updating the channel selection multiplexer on the AIM-16 expansion card, or the AIO8 multiplexer.

Utility Software

Utility software included with the AIO8 is provided in MS-DOS format on CD. Two menu-driven setup programs, a driver in linkable object form and binary form, and sample programs are provided. One setup program is a configuration and calibration tool for the AIO8, and the second setup program is used when an AIM-16 and/or LVDT-8 is used in conjunction with the AIO8. In the case of the AIM-16, gains are assignable on a channel-by-channel basis. Linearization for all the commonly used thermocouple types as well as for platinum RTDs is also menu selectable.

A driver configuration file is generated or modified by the setup program and can be used to configure the AIO8 driver. System Software describes the format of this configuration file. This driver has 17 Tasks as will be described in detail in AIO8 Driver Reference of this manual.

Sample programs are provided in BASIC, QuickBASIC, Pascal and "C".

Enhancements

Capabilities of the AIO8 can be greatly enhanced by use of one or more of the following hardware devices or software packages:

a. STA-37 Screw Terminal Cards.

These terminal boards allow direct input/output signal connections via screw terminals.

The card provides a breadboard area with ±12V and +5V computer power. This breadboard area can be used for amplifiers, filters, and other user-assembled circuits.

b. AIM-16 Expansion Multiplexer and Instrumentation Amplifier

The AIM-16 is a 16-channel amplifier/multiplexer that features differential-input capability and a choice of either DIP switch selectable gains or software programmable gains. The AIM-16 allows multiplexing of 16 analog input signals to a single AIO8 analog input channel. As described earlier, up to eight AIM-16's can be connected to a single AIO8 to provide input capability for up to 128 analog inputs.

The AIM-16 includes a low-drift instrumentation amplifier with DIP switch selectable gains of 0.5, 1, 2, 5, 10, 25, 50, 100, 200, 500, and 1000. In addition, these gains can be software programmed to provide individual gains on a channel-by-channel basis.

For thermocouple measurements, a cold junction sensor is provided to allow reference junction compensation, via software, for thermocouple inputs. The reference junction output may be assigned to channel 0 of the AIM-16 or, alternatively, may be jumpered to an unused AD12-8 input channel. Open-thermocouple or "break detect" circuitry is provided.

The AIM-16 may also be used with 3-wire RTD's(AIM-16R), strain gages, and 4-20mA current transmitter inputs. In this latter case, an application-specific version, the AIM-16I, is available.

c. LVDT-8 Multiplexer and Interface Card

This card provides AC excitation and signal conditioning to eight independent LVDT transducers. As many as eight LVDT-8's may be connected to an AIO8 to accommodate up to 64 transducers.

d. EASY ACCESS

EASY ACCESS is a real-time, menu-driven, data acquisition software package that takes signals from measurement devices and enters them into the computer without need for you to do any application programming.

EASY ACCESS supports up to 128 analog and 120 digital sensor inputs and also gives you the ability to define up to 80 additional "calculated channels". Calculated channels are software channels derived from mathematical computations performed on sensor input data or on other calculated channels.

Data files generated by EASY ACCESS can be formatted for direct link to LOTUS 1-2-3 or other analysis packages, thus providing full analytical power and graphing functions for data reduction. EASY ACCESS is suggested to anyone involved in repetitive data recording, or anyone who wishes to minimize their programming involvement when using the AIO8.

e. LABTECH NOTEBOOK

LABTECH NOTEBOOK is a menu-driven data acquisition software package. It is capable of foreground or background operation and, although more costly than EASY ACCESS, provides more capabilities including ICON setup, variable sample rates on a channel-by-channel basis, additional mathematical and statistical calculations, and compatibility with expanded memory. DOS and Windows versions are available.

Specifications

Analog Inputs

- Channels: 8 single-ended inputs with common ground.
- Voltage Range: ±5V.
- Resolution: 12 binary bits.
- Accuracy: ±0.01% of reading ±1 LSB.
- Input Impedance: 10MΩ or 100nA at 25 °C.
- Overvoltage: ±30VDC.
- Linearity: ±1 LSB.
- Throughput: 4,000 conversions per second maximum.
- Temp. Coefficient: ±10 μV/ °C. zero stability, ±25 μV/ °C. gain stability.
- Common Mode Rejection: (when used with AIM-16)
- 90 dB when gain = 1
- 125 dB when gain = 100

Reference Voltage Output

- Voltage: 10.0VDC ±0.1VDC at up to 2mA.

Digital I/O

- Inputs:
  - Logic high: 2.4 to 5.0 VDC at 20μA source current.
  - Logic low: 0 to 0.4 VDC at -0.8mA sink current.
- Outputs:
  - Logic high: 2.4V to 5.0V at 0.4mA source current.
  - Logic low: 0V to 0.4V at 8mA sink current.

Interrupt Channel

- Levels: Levels 2 through 7, jumper selectable.
- Enable: Via software.
- Source: External input.

Programmable Timer

- Type: 8254 programmable interval timer.
- Counters: Three 16-bit down counters.
- Output Drive: 2.2mA at 0.45V (5 LSTTL loads).
- Input Load: ±10µA, TTL/DTL/CMOS compatible, gate and clock.
- Clock Frequency: DC to 8MHz.
- Active Count Edge: Negative edge.
- Min Clock Pulse Width: 50nS high/50nS low.
- Timer Range: 2.5 MHZ to <1 pulse/hr.

Environmental

- Operating Temp: 0 °C. to 60 °C.
- Storage Temp: -40 °C. to 100 °C.
- Humidity: 0 to 90% RH, non-condensing.
- Size: 7.0 inches long (178 mm), requires full-size slot.
- Power Required: +5VDC: 320mA maximum
  +12VDC: 10mA maximum
  -12VDC: 15mA maximum

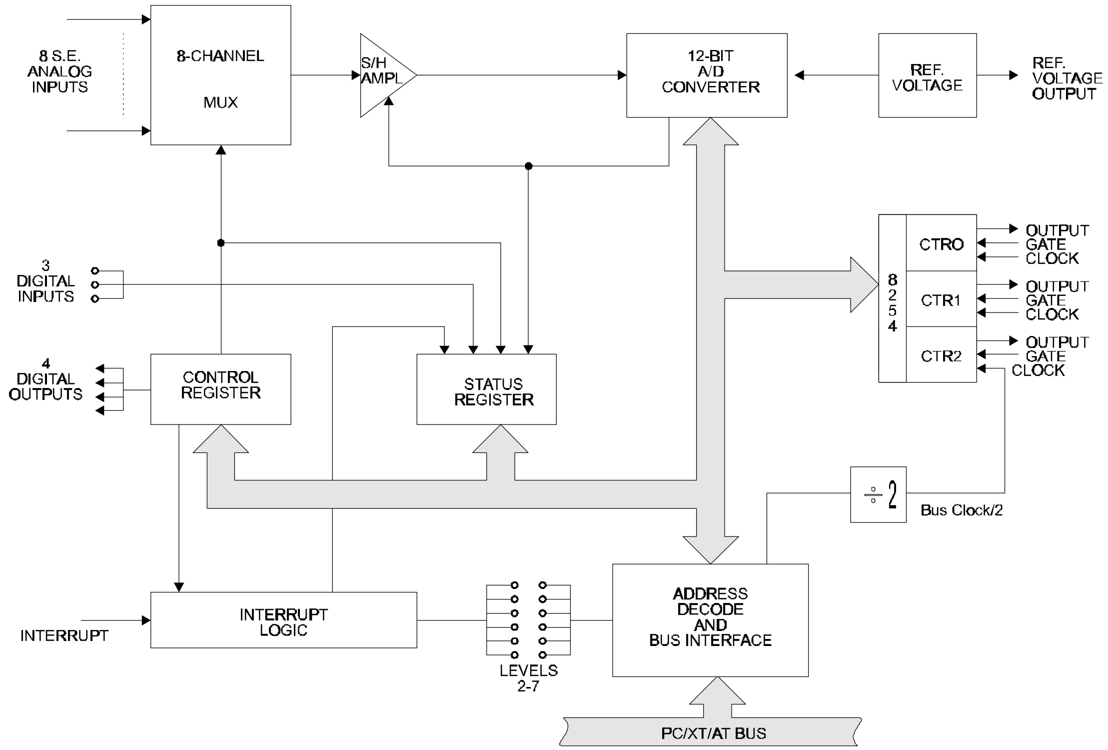

Figure 1-1: AIO8 Block Diagram

Chapter 2: Installation

The software provided with this card is contained on either one CD or multiple diskettes and must be installed onto your hard disk prior to use. To do this, perform the following steps as appropriate for your software format and operating system. Substitute the appropriate drive letter for your CD-ROM or disk drive where you see d: or a: respectively in the examples below.

CD Installation

DOS/WIN3.x

1. Place the CD into your CD-ROM drive.
2. Type d:K to change the active drive to the CD-ROM drive.
3. Type installK to run the install program.
4. Follow the on-screen prompts to install the software for this card.

WIN95/98/NT

a. Place the CD into your CD-ROM drive.
b. The CD should automatically run the install program after 30 seconds. If the install program does not run, click START | RUN and type d:install, click OK or press K.
c. Follow the on-screen prompts to install the software for this card.

3.5-Inch Diskette Installation

As with any software package, you should make backup copies for everyday use and store your original master diskettes in a safe location. The easiest way to make a backup copy is to use the DOS DISKCOPY utility.

In a single-drive system, the command is:

diskcopy a: a:K

You will need to swap disks as requested by the system.
In a two-disk system, the command is:

diskcopy a: b:K

This will copy the contents of the master disk in drive A to the backup disk in drive B.

To copy the files on the master diskette to your hard disk, perform the following steps.

a. Place the master diskette into a floppy drive.

b. Change the active drive to the drive that has the diskette installed. For example, if the diskette is in drive A, type a:K.

c. Type installK and follow the on-screen prompts.

Directories Created on the Hard Disk

The installation process will create several directories on your hard disk. If you accept the installation defaults, the following structure will exist.

[CARDNAME]

Root or base directory containing the SETUP.EXE setup program used to help you configure jumpers and calibrate the card.

DOS\PSAMPLES: A subdirectory of [CARDNAME] that contains Pascal samples.
DOS\CSAMPLES: A subdirectory of [CARDNAME] that contains "C" samples.
Win32\language: Subdirectories containing samples for Win95/98 and NT.

WinRisc.exe

A Windows dumb-terminal type communication program designed for RS422/485 operation. Used primarily with Remote Data Acquisition Pods and our RS422/485 serial communication product line. Can be used to say hello to an installed modem.

ACCES32

This directory contains the Windows 95/98/NT driver used to provide access to the hardware registers when writing 32-bit Windows software. Several samples are provided in a variety of languages to demonstrate how to use this driver. The DLL provides four functions (InPortB, OutPortB, InPort, and OutPort) to access the hardware.

This directory also contains the device driver for Windows NT, ACCESNT.SYS. This device driver provides register-level hardware access in Windows NT. Two methods of using the driver are available, through ACCES32.DLL (recommended) and through the DeviceIOControl handles provided by ACCESNT.SYS (slightly faster).

SAMPLES

Samples for using ACCES32.DLL are provided in this directory. Using this DLL not only makes the hardware programming easier (MUCH easier), but also one source file can be used for both Windows 95/98 and WindowsNT. One executable can run under both operating systems and still have full access to the hardware registers. The DLL is used exactly like any other DLL, so it is compatible with any language capable of using 32-bit DLLs. Consult the manuals provided with your language's compiler for information on using DLLs in your specific environment.

VBACCES

This directory contains sixteen-bit DLL drivers for use with VisualBASIC 3.0 and Windows 3.1 only. These drivers provide four functions, similar to the ACCES32.DLL. However, this DLL is only compatible with 16-bit executables. Migration from 16-bit to 32-bit is simplified because of the similarity between VBACCES and ACCES32.

PCI

This directory contains PCI-bus specific programs and information. If you are not using a PCI card, this directory will not be installed.

SOURCE

A utility program is provided with source code you can use to determine allocated resources at run-time from your own programs in DOS.

PCIFind.exe

A utility for DOS and Windows to determine what base addresses and IRQs are allocated to installed PCI cards. This program runs two versions, depending on the operating system. Windows 95/98/NT displays a GUI interface, and modifies the registry. When run from DOS or Windows3.x, a text interface is used. For information about the format of the registry key, consult the card-specific samples provided with the hardware. In Windows NT, NTioPCI.SYS runs each time the computer is booted, thereby refreshing the registry as PCI hardware is added or removed. In Windows 95/98/NT PCIFind.EXE places itself in the boot-sequence of the OS to refresh the registry on each power-up.

This program also provides some COM configuration when used with PCI COM ports. Specifically, it will configure compatible COM cards for IRQ sharing and multiple port issues.

WIN32IRQ

This directory provides a generic interface for IRQ handling in Windows 95/98/NT. Source code is provided for the driver, greatly simplifying the creation of custom drivers for specific needs. Samples are provided to demonstrate the use of the generic driver. Note that the use of IRQs in near-real-time data acquisition programs requires multi-threaded application programming techniques and must be considered an intermediate to advanced programming topic. Delphi, C++ Builder, and Visual C++ samples are provided.

Findbase.exe

DOS utility to determine an available base address for ISA bus , non-Plug-n-Play cards. Run this program once, before the hardware is installed in the computer, to determine an available address to give the card. Once the address has been determined, run the setup program provided with the hardware to see instructions on setting the address switch and various option selections.

Poly.exe

A generic utility to convert a table of data into an nth order polynomial. Useful for calculating linearization polynomial coefficients for thermocouples and other non-linear sensors.

Risc.bat

A batch file demonstrating the command line parameters of RISCTerm.exe.

RISCTerm.exe

A dumb-terminal type communication program designed for RS422/485 operation. Used primarily with Remote Data Acquisition Pods and our RS422/485 serial communication product line. Can be used to say hello to an installed modem. RISCTerm stands for Really Incredibly Simple Communications TERMinal.

Configuration File

The Configuration File has several purposes; most of which are associated with use of the software drivers provided with your AIO8 card. These are as follows:

a. Provide means to automatically configure the driver and, thus, avoid need for multiple calls to the driver to do the setup.
b. Allow the setup programs to do the work of configuring the driver. When you use the setup programs to assist in configuring the card, this information is saved in the configuration file and can then be used by the drivers.
c. When you use the EASY ACCESS software package, that software uses the configuration file to configure itself.

The Configuration File, SETUP.CFG, is generated or modified with the setup programs. It can also be generated or modified by an editor or a word processor in the non-document mode. This file is a structured file containing setup information for:

a. AIO8 plug in PC card.
b. AIM-16 sub-multiplexer card.
c. LVDT-8 sub-multiplexer card.
d. Programmable gain assignments.
e. Curve assignments.

The configuration file must contain 12 lines of information or data. The file is strictly text only (ASCII). Each line of information is made up of a description field, an equal sign (=), and a setup information field. Each line in the configuration file supplies data for a specific parameter and must be in the correct order. Table 2-1 contains an example of a configuration file.

If you use programmable gains on the AIM-16, the counters will not be available for general use. The counters are required by the drivers to set the gain on the AIM-16.

Base Address	$300
A/D Channel 0	1:F:Tt3UUUUUUUUSSSSS
A/D Channel 1	2:UUUUUUUU
A/D Channel 2	0:U
A/D Channel 3	0:S
A/D Channel 4	0:S
A/D Channel 5	0:S
A/D Channel 6	0:S
A/D Channel 7	0:S
Voltage Range	5
Bipolar/Unipolar	B
IRQ Channel	3

Table 2-1: Configuration File Example

Base Address

Values may be entered as a decimal string, a hexadecimal string, or as a binary string. A decimal string is any string of digits made up of the digits 0 through 9 as follows:

Decimal format ... DDDDD (e.g. 768)

Hexadecimal strings consist of a string of digits containing the digits 0 through 9 and the letters A through F. Hexadecimal strings may be made up in one of two ways; Pascal or "C" as follows:

Pascal format .... $HHHH (e.g. $300)
"C" format .... 0xHHHH (e.g. 0x300)

Binary strings are made up of 0's and 1's preceded by a # as follows:

Binary format ... #bbbbbbbbbb (e.g. #1100000000)

MUX Extensions

The mux channel extension to the A/D board can be made up of an AIM-16 description string, an LVDT-8 description string, or a raw channel description string. The first character following the "=" is a card code that defines which description string follows.

Card Code 0

Raw A/D channel; i.e., no external sub-multiplexer is attached. The 0: may be followed by one of three characters; T, U, or S.

T signifies that a reference junction on a sub-multiplexer card is directly attached to the channel.
S signifies that the channel is skipped; i.e., not used.
U signifies that the channel is unskipped; i.e., treated as a normal A/D input.

Card Code 1

An AIM-16 sub-multiplexer is attached. Immediately following the 1: card code will be a unit-of-measure code. Possible codes are F (units are degrees Fahrenheit), C (units are degrees Celsius), and N (no units specified). Note: If N is used but a thermocouple is installed, then the default is degrees F.

Following the unit-of-measure code is a series of 16 channel-code letters or numbers that determine how the channel is to be used. Any of the following codes may be used for each channel:

S:	The channel is skipped; i.e., not used.
U:	The channel is used (gain code 0).
T:	The channel is used for a thermocouple reference junction
t,k,j,e,r,s,b,a,u:	The channel is used with the indicated thermocouple type or platinum RTD attached. ("a" is used for RTDs with an alpha of 392 and "u" is used for RTD's with an alpha of 385.)
A:	The channel is to have automatic gain ranging. Note that the AIM-16 gain switches must be set for programmable gain for automatic gain ranging to work properly.
0,1,2,3,4,5,6,7:	The channel is to be set to the indicated gain. Gains associated with these code numbers are defined in the gain table contained in TASK 4's reference in AD12-8 DRIVER REFERENCE. The AIM-16 gain switches must be set for programmable gain for this function to work properly.

Card Code 2

An LVDT-8 sub-multiplexer card is attached. Following the card code is a series of eight channel-code letters that define how the channels are to be used. The letters are U for unskipped and S for skipped.

Voltage Range

Values indicate the voltage range that has been selected by jumpers on the card. The only possible value for the AIO8 is 5.

Bipolar/Unipolar Mode

A "B" signifies that the card is set to bipolar mode while a "U" indicates that the card is set to the unipolar mode. The only value allowed for the AIO8 is "B".

IRQ Level

Values indicate which IRQ interrupt level will be used. Allowable values are 2 through 7. If the setup program is used, it will put a 0 on this line if no interrupt level is selected. If the driver detects a 0 on this line, it installs a default of IRQ3.

Example

Using the configuration file listed in Table 2-1: Configuration File Example, the information has the following meaning:

a. The base address is set to hex 300 in Pascal format.
b. Channel 0 of the AIO8 has an AIM-16 sub-multiplexer card attached. The unit of measure for this card is oF. The AIM-16 channel 0 is used for reference junction, channel 1 is a "t" thermocouple type input, channel 2 has a gain code of 3, channels 3 through 10 are used (gain code defaults to zero), and the remaining channels are skipped (unused).
c. Channel 1 of the AIO8 has an LVDT-8 attached, with all eight channels unskipped.
d. Channel 2 of the AIO8 is a direct channel that is unskipped.
e. All other A/D channels are direct and skipped

Chapter 3: Hardware Configuration and Installation

Option Selection

The AIO8 card features are selected by hardware jumper. At least one of each of the option categories must be selected if the card is to operate correctly. The setup program provided with the card provides menu-driven pictorial presentations to help you quickly set up the card. You may also refer to Figure 3-1: Option Selection Map, and the following sections to set up the card. The card should not be plugged into the computer at this time.

Interrupts

Interrupts originating from an external source (pin 24) are supported. Interrupts are enabled by setting the IEN bit of Control Register #1 high. Selection of the interrupt source is made by jumper. Either an External source (positive edge trigger) connected via I/O connector Pin 24 or the EOC signal from the A/D converter are selected by placement of the EOC/EXT jumper on the card. Interrupt levels IRQ2 through IRQ7 are available. The desired level is selected by installing a jumper in one of the jumper locations marked IRQ2 through IRQ7.

Selecting a Base Address

You need to select an unused segment of eight consecutive I/O addresses. The base address will be the first address in this segment. The base address may be selected anywhere on a 8-byte boundary within the I/O address range 100-3FF hex providing that it does not overlap with other functions. The following procedure will show you how to select the base I/O address.

1. Check Table 3-1: Standard Base Address Assignments for a list of standard address assignments and then check what addresses are used by any other I/O peripherals that are installed in your computer. (Memory addressing is separate from I/O addressing, so there is no possible conflict with any add-on memory that may be installed in your computer. We urge that you carefully review the address assignment table before selecting a card address. If the addresses of two installed functions overlap, unpredictable computer behavior will result.
2. From this list, select an unused portion of eight consecutive I/O address. Note from Table 3-1 that the sections 280-2EF and 330-36F are unused. This address space is good area to select a base address from. Also, if you are not using a given device in Table 3-1, then you may use that base address as well. For example, most computers do not have a prototype card installed. If your computer does not have one, then base address 300 hex is a good choice for a base address.
3. Finally make sure that the base address you have chosen has the last digit as 0 or 8. This insures that your base address is on an 8-byte boundary.

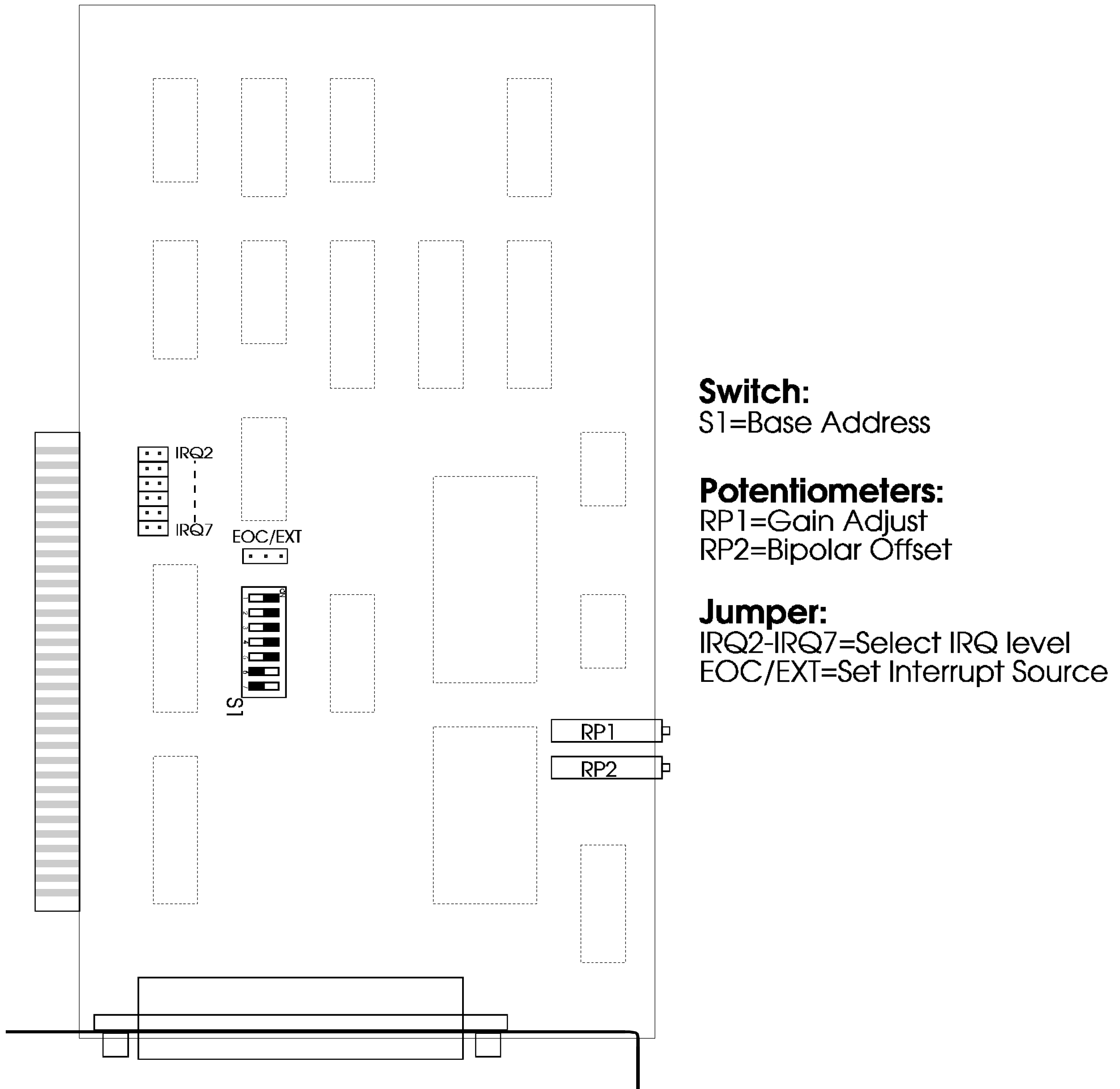

Figure 3-1: Option Selection Map

Hex Range	Usage
000-01F	DMA Controller 1
020-03F	INT Controller 1, Master
040-05F	Timer
060-06F	8042 Keyboard
070-07F	Real Time Clock, NMI Mask
080-09F	DMA Page Register
0A0-0BF	INT Controller 2
0C0-0DF	DMA Controller 2
0F0	Clear Math Coprocessor Busy
0F1	Reset Coprocessor
0F8-0FF	Arithmetic Processor
1F0-1F8	Fixed Disk
200-207	Game I/O
278-27F	Parallel Printer Port 2
2F8-2FF	Asynchronous Comm'n (Secondary)
300-31F	Prototype Card
360-36F	Reserved
378-37F	Parallel Printer Port 1
380-38F	SDLC or Binary Synchronous Comm'n 2
3A0-3AF	Binary Synchronous Comm'n 1
3B0-3BF	Monochrome Display/Printer
3C0-3CE	Local Area Network
3D0-3DF	Color/Graphic Monitor
3F0-3F7	Floppy Diskette Controller
3F8-3FF	Asynchronous Comm'n (Primary)

Table 3-1: Standard Address Assignments for 286/386/486 Computers

Setting the Base Address

The AIO8 base address is selected by DIP switch S1 located in the lower right hand portion of the card. Switch S1 controls address bits A3 through A9. Bits A0 through A2 are used for the eight address locations in I/O space required by the AD12-8. The following procedure will show you how to set the base address. See Table 3-2: Base Address Example below, for a graphic representation of this example.

1. We will use base address 300 hex as an example. Determine the binary representation for your base address. In our example, 300, the binary representation is 11 0000 0000. The conversion multipliers for each binary bit are contained in Table 3-2 for reference.
2. Locate switch S1 on the lower right side of the card. Note there are seven switches, which will be used to set the seven most-significant bits in the binary representation from step 1).
3. Note from Table 3-2 that switch position A9 corresponds to the most significant bit in your binary representation. For each bit in your binary representation, if the bit is a one, turn the corresponding switch off; if the bit is zero, turn the corresponding switch on.

Hex representation	**3**		**0**				**0**
Binary representation	1	1	0	0	0	0	0
Conversion multiplier	2	1	8	4	2	1	8
Switch ID	A9	A8	A7	A6	A5	A4	A3
Switch setting	OFF	OFF	ON	ON	ON	ON	ON

Table 3-2: Base Address Example

Using the Setup Program to Set the Base Address

The setup program provided with AIO8 contains an interactive menu-driven program to assist you in setting the base address. The following procedure demonstrates the use of the setup program.

1. Choose a desired base address.
2. Execute the setup program by typing SETUP.EXE and pressing the ENTER key.
3. Select the first item in the menu, 1) Set board address. with the up or down arrow key and press ENTER.
4. Enter the desired base address in hex, the program will display a graphic representation of how you should set the switches. You may press the space bar to try another address.
5. Set DIP switch S1 as shown on the graphic representation.

Installing the Card

The following procedure will show you how to install the AIO8 inside the computer.

1. Ensure that all options have been set as described in the first part of this section. Be sure to pay close attention to base address selection.
2. Turn off the power switch of your computer and remove the power cords from the wall outlet.

Caution

Failure to Remove Power from the Computer Could Result in Electrical Shock, or Damage to Your Computer System.

3. Remove the computer cover.
4. Locate an unused full length slot, and remove the blank I/O back plate.
5. Insert the card in the slot, and install the I/O back plate screw. Make sure that the card mounting bracket is properly screwed into place and that there is a positive chassis ground.
6. Inspect the installation for proper fit and seating of the card.
7. Replace the computer cover.
8. Reapply power to the computer.
9. Perform the calibration procedure which follows.

A 37-pin connector is provided on the mounting bracket for input/output connections. To assure that there is minimum susceptibility to EMI and minimum radiation, it is important that the card mounting bracket be properly screwed into place and that there be a positive chassis ground. Also, proper EMI cabling techniques (cable connect to chassis ground, shielded twisted-pair wiring, etc) must be used for input and output wiring.

CE-marked versions of AIO8 meet the requirements of EN50081-1:1992 (Emissions), EN50082-1:1992(Immunity), and EN60950 (Safety).

Calibration and Test

All cards are calibrated prior to shipment. However, periodic calibration of AIO8 is recommended to retain full accuracy. The calibration interval depends to a large extent on the type of service that the card is subjected to. For environments where there are frequent large changes of temperature and/or vibration, a three-month interval is suggested. For laboratory or office conditions, six months to a year is acceptable.

A 4-1/2 digit digital multimeter is required as a minimum to perform satisfactory calibration. Also, a voltage calibrator or a stable noise-free DC voltage source that can be used in conjunction with the digital multimeter is required.

Calibration is performed using the SETUP program supplied with your card. This program will lead you through the set up and calibration procedure with prompts and graphic displays that show the settings and adjustment trim pots. This calibration program also serves as a useful test of the AIO8 A/D functions and can aid in troubleshooting if problems arise.

Calibration Procedure

The following procedure is brief and is intended for use in conjunction with the calibration part of the setup program.

1. Start the calibration program by typing AIO-8PST and press the ENTER key at the DOS prompt.
2. Use the relevant menu selections to set the switches and jumper for the manner in which the card will be used; i.e., base address and IRQ. These settings are used by the calibration portion of the program.
3. Use the arrow key to select option 7) Calibrate, then press the ENTER key.
4. Use the arrow key to select option 1. Set Offset, from the Action Menu at the top left hand corner of the screen.
5. Following the instructions on the screen, perform the offset adjustment
6. Use the arrow key to select option 2. Set Gain, from the Action Menu.
7. Following the instructions on the screen, perform the gain adjustment
8. Use the arrow key to select option 3. Check, from the Action Menu.
9. Following the instructions on the screen, perform the calibration check.
10. This completes the calibration procedure.

Chapter 4: Programming

This section provides you with information on how to program the AIO8. First, information is provided on how to program the card using direct register access. Following this is information on using the drivers provided with the AIO8. If you plan to use the driver provided, refer to the section in this chapter, Programming Using the Driver, and to AIO8 Driver Reference. The following section, Register Definitions, is informational only for those of you planning to use the driver, but may be useful to gain an understanding of how the card functions.

At the lowest level, the AIO8 can be programmed using direct I/O input and output instructions. In BASICA, these are the INP (X) and OUT X,Y functions. Assembly language and most high level languages have equivalent instructions. Use of these functions usually involves formatting data and dealing with absolute I/O addresses. Although not demanding, this can require many lines of code and requires an understanding of the devices, data format, and architecture of the AIO8. You may find it easer to design your program using the supplied drivers.

AIO8 Register Address Map

The AIO8 uses eight consecutive addresses in I/O space as follows:

Register Address	Read Function	Write Function
BASE ADDRESS + 0	A/D Low Byte	Start 8-bit Conversion
BASE ADDRESS + 1	A/D High Byte	Start 12-bit Conversion
BASE ADDRESS + 2	Status Register	Control Register
BASE ADDRESS + 3	Not Used	Not Used
BASE ADDRESS + 4	Read Counter #0	Load Counter #0
BASE ADDRESS + 5	Read Counter #1	Load Counter #1
BASE ADDRESS + 6	Read Counter #2	Load Counter #2
BASE ADDRESS + 7	Not Used	Counter Control

Table 4-1: AIO8 Register Address Map

Register Definitions

Control Register

Base + 2 Write: Read or write the control register.

B7	B6	B5	B4	B3	B2	B1	B0
OP3	OP2	OP1	OP0	IEN	MA2	MA1	MA0

0P0-OP3: These bits correspond to four general-purpose digital output lines. These lines can be used for external control functions such as selecting inputs from the AIM-16, or LVDT-8 sub-multiplexer cards.

IEN: This bit enables/disables AIO8 external interrupts. 1 = enabled, 0 = disabled. When enabled, external interrupts from pin 24 of the I/O connector are passed through on the selected IRQ level.

MA0-MA2: These bits select the analog multiplexer channel address on the AIO8 card (Channels 0 through 7).

Status Register

The Status register provides information about the operation of the card.

Base + 2 Read: Read the card status.

B7	B6	B5	B4	B3	B2	B1	B0
EOC	IP3	IP2	IP1	IRQ	MA2	MA1	MA0

EOC: End of conversion. If EOC = 1, an A/D conversion is underway. If EOC is 0, then the A/D data registers contain valid data from the previous conversion and the A/D is ready to perform the next conversion.

IP1-IP3: These bits correspond to three general purpose digital input lines. They may be used for any digital data input.

IRQ: After generation of an interrupt, the AIO8 card sets this bit high(1). It is reset to state 0 by a computer reset, or a write to the control register.

MA2-MA0: These bits define the analog multiplexer channel address on the AD12-8 card (channels 0 through 7).

A/D Registers

A/D data are in true binary form and are latched in the A/D registers at the end of each conversion. These are read at BASE ADDRESS and BASE ADDRESS +1 in low-byte/high-byte sequence. The data are available until the end of the next A/D conversion.

Base + 0 Read: Contains the lower four bits of the A/D converter output in binary form.

B7	B6	B5	B4	B3	B2	B1	B0
AD3	AD2	AD1	AD0	0	0	0	0

AD0-AD3: The lower four bits of the A/D conversion, AD0 is the least-significant bit.
B0-B3: These bits are always 0.

Base + 1 Read: Contains the upper twelve bits of the A/D converter output in binary form.

B7	B6	B5	B4	B3	B2	B1	B0
AD11	AD10	AD9	AD8	AD7	AD6	AD5	AD4

AD4-AD11: The upper twelve bits of the A/D conversion, AD4 is the least-significant bit.

The A/D data bits are offset-binary coded. The following table demonstrates the binary encoding used.

Binary	Hex	Analog Input Voltage
0000 0000 0000	000	-5.0000V (-Full Scale)
0000 0000 0001	001	-4.9976
•	•	•
•	•	•
0100 0000 0000	400	-2.5V (- ½ Scale)
•	•	•
•	•	•
1000 0000 0000	800	±0V
1000 0000 0001	801	+0.0024V
•	•	•
•	•	•
1100 0000 0000	C00	+2.5000V (+½ Scale)
•	•	•
•	•	•
1111 1111 1111	FFF	+4.9976 (+Full Scale

Base + 0 Write: A write to this location starts an 8-bit A/D conversion. The data written is irrelevant. This causes the EOC bit of the status register to go high until the conversion is complete.

Base + 1 Write: A write to this location starts a 12-bit A/D conversion. The data written is irrelevant. This causes the EOC bit of the status register to go high until the conversion is complete.

Counter/Timer Registers

Base + 4 Write/Read: Counter #0 read or write. When writing, this register is used to load a counter value into the counter. The transfer is either a single or double byte transfer, depending on the control byte written to the counter control register at BASE ADDRESS + 7. If a double byte transfer is used, then the least-significant byte of the 16 bit value is written first, followed by the most significant byte. When reading, the current count of the counter is read. The type of transfer is also set by the control byte.

Additional information about the type 8254 counters is presented in PROGRAMMABLE INTERVAL TIMER section of this manual. However, for a full description of features of this extremely versatile IC, refer to the Intel 8254 data sheet.

Base + 5 Write/Read: Counter #1 read or write. See description for Base + 4 Write/Read.

Base + 6 Write/Read: Counter #2 read or write. See description for Base + 4 Write/Read.

Base + 7 Write: The counters are programmed by writing a control byte into a counter control register at BASE ADDRESS + 7. The control byte specifies the counter to be programmed, the counter mode, the type of read/write operation, and the modulus. The control byte format is as follows:

B7	B6	B5	B4	B3	B2	B1	B0
SC1	SC0	RW1	RW0	M2	M1	M0	BCD

SC0-SC1: These bits select the counter that the control type is destined for.

SC1	SC0	Function
0	0	Program Counter 0
0	1	Program Counter 1
1	0	Program Counter 2
1	1	Read Back Command

RW0-RW1: These bits select the read/write mode of the selected counter.

RW1	RW0	Counter Read/Write Function
0	0	Counter Latch Command
0	1	Read/Write LS Byte
1	0	Read/Write MS Byte
1	1	Read/Write LS Byte, then MS Byte

M0-M2: These bits set the operational mode of the selected counter.

Mode	M2	M1	M0
0	0	0	0
1	0	0	1
2	X	1	0
3	X	1	1
4	1	0	0
5	1	0	1

BCD: Set the selected counter to count in binary (BCD bit = 0) or BCD (BCD bit = 1).

Programming Using the Driver

Using direct register access to program the AIO8 is straightforward but the coding can be rather tedious. To assist you in building your application quickly, a driver is provided. This driver is provided in three forms. Which form you use will depend on the programming language that you intend to use in your application. A task reference for this driver is provided in AIO8 Driver Reference. The driver file names and their language use are as follows:

AIO8DRV.BIN A BASIC loadable driver for use with most interpreted BASIC languages.
AIO8DRV.OBJ A Pascal and QuickBASIC linkable driver in object form.
AIO8DRVC.OBJ A "C" linkable driver in object form.

Also, to help you understand how to use the driver with your program, sample programs are provided in three languages; "C", Pascal, and QuickBASIC.

SAMPLE 1 Demonstrates data acquisition using polling, with an AIM-16 in programable gain mode.
SAMPLE 2 Demonstrates timer-driven data acquisition using interrupts.
SAMPLE 3 Same as Sample 1 but uses the configuration file to set up the driver.

To access the functions of the driver, a call to a single procedure within the driver is used. The name of the procedure for the driver is AIO8DRV. The procedure is called with three variables, which are defined as follows:

task: The number of the task to perform. A reference with a list of tasks for each driver are provided in AIO8 Driver Reference.

parameters: This is an array of integers which contains information required by the driver. AIO8 Driver Reference defines what values need to be passed for each task. The array should hold five integers.

status: An error code is returned in this variable. A zero is returned if there is no error.

When calling the procedure, certain important requirements must be met:

a. The three variables must be declared as global. When variables are declared global, the data segment register of the processor will contain the segment of these variables, and the driver is designed to use this segment. If variables are declared locally, the stack is the segment for the variables. The driver would still use the data segment, which would cause it to write or read data in the wrong area of memory.

b. The driver expects parameters to be integer type variables and will write to and read from the variables on this assumption. The driver will not function properly if non-integer variables are used in the call.

c. The variables should be passed by reference. The driver expects offsets of the variables so that data may be returned when required.

d. The passed variables are positional. That is, the variables must be specified in the sequence (task, parameters, status). Their location is derived sequentially from the variable pointers on the stack.

e. The driver will not function properly if arithmetic functions (+, -, x, etc) are specified within the variable list bracket.

Using the Driver with Turbo or Borland C

The following list shows you how to use the driver with Borland or Turbo C. You may refer to any of the "C" example programs for further illustration.

A. Include the AIO8DRVC.h header in your program. This simple header provides a function prototype of the procedure call.

```
#include "aio8drvc.h"
```

B. Declare the three variables for the driver globally, at the beginning of your program.

```
inttask,params[5],status;
```

C. Make your assignment to these variables as desired for the function you wish to perform. See Driver Reference for details on each task.

D. Make the call to the driver, passing the offset of each parameter.

```
aio8drv(FP_OFF(&task),FP_OFF(params),FP_OFF(status));
```

E. Create a project file within the Turbo C environment, and add the name of your program with the .C extension, and the name of the driver with a .OBJ extension.

F. Select "LARGE" memory model under the compiler section of the options menu.

G. Compile and link the program.

Using the Driver with Microsoft C

To use the driver with Microsoft C version 6.0, add the following code to your application code as shown below:

```
_asm
  {
    push DS
    mov AX,ES
    mov DS,AX
  }
/* call driver as normal */
aio8drv(FP_OFF(&task),FP_OFF(params),FP_OFF(status));

_asm
  {
    pop DS
  }
```

If you are using a version of Microsoft C prior to version 6.0 use the following code:

```
_asm_emit 0x1E
_asm_emit 0x8E
_asm_emit 0xD8
/* call driver as normal */
aio8drv(FP_OFF(&task),FP_OFF(params),FP_OFF(status));

_asm_emit 0x1F
```

These changes work around a peculiarity of Microsoft C, enabling our drivers to locate the variables used in the program.

Using the Driver with Turbo Pascal

The following procedure will show you how to use the driver with Borland Turbo Pascal. You may refer to any of the Pascal example programs for further illustration.

A. Include the following compiler directive at the beginning of your program.

```
{$L aio8drv}
```

B. Declare the three variables for the driver globally, at the beginning of the program.

```
type param_array = array[1..5] of integer;
var params      : param_array;
    task,status : integer;
```

C. Declare the driver function as external in using a prototype declaration

```
procedure aio8drv(task:word; param:word; status:word);external;
```

D. Make your assignment to these variables as desired for the function you wish to perform. See AIO8 Driver Reference for details on each task.

E. Make the call to the driver.

```
aio8drv(ofs(task),ofs(params),ofs(status));
```

F. Compile and link the program.

Using the Driver with QuickBASIC

The following procedure will show you how to use the driver with Microsoft QuickBASIC. You may refer to any of the QuickBASIC sample programs for further illustration. The following procedure will allow you to use the driver both in the QuickBASIC environment and from the command line compiler.

A. Declare the three variables for the driver as global.

```
DIM TASK%, STAT%, PARAMS%(5)
```

B. The array dimension statement must be followed by the COMMON SHARED statement for the driver to be able to find the array. Note: Steps A and B are necessary for any array that will be used by the driver. Certain tasks within the driver require the address of a data buffer, so these two steps would need to be performed for those arrays as well.

```
COMMON SHARED PARAM%()
```

C. Now DECLARE the driver routine. This declaration must include a BYVAL statement before the array variable.

```
DECLARE SUB AIO8DRV(TASK%, BYVAL PARAM%, STAT%)
```

D. Make your assignment to these variables as desired for the function you wish to perform. See AIO8 Driver Reference for details on each task.

E. Make the call to the driver. The CALL statement must explicitly pass the offset of the array variable.

```
CALL AIO8DRV(TASK%, VARPTR(PARAM%(1)), STAT%)
```

F. To use the program and driver in the environment, you must link a Quick Library. Perform the following command from the command line.

```
LINK /Q AIO8DRV.OBJ,AIO8DRV.QLB,,BQLB45.LIB; [ENTER]
```

G. Now load the Quick Library when starting the environment.

```
QB /L AIO8DRV.QLB [ENTER]
```

H. Use the start command from the run menu to execute the program.

I. To prepare an EXE file from the command line, use the following compile and link commands.

```
BC /o YOURPROG;[ENTER]
LINK YOURPROG+AIO8DRV;[ENTER]
```

Using the Driver with BASIC

The following procedure will show you how to use the driver with most BASIC languages. You may refer to any of the BASIC sample programs for further illustration.

A. Declare the three variables for the driver as global.

```
10      DIM TASK%, STAT%, PARAMS(5)
```

B. Define a segment within memory to load the driver. You must make sure this segment is not used by BASIC or your program. You may do this by estimating the amount of memory used by your program and the BASIC interpreter you are using, then choosing a segment well above this area.

```
20      DRIVERSEG = &H5000 30
30      DEF SEG = DRIVERSEG
```

C. Load the driver into memory starting at offset 0 within the defined segment.

```
40      DRIVER = 0
50      BLOAD "aio8drv.bin",DRIVER
```

D. Make your assignment to these variables as desired for the function you wish to perform. See AIO8 Driver Reference for details on each task.

E. Make the call to the driver.

```
60      CALL aio8drv(TASK%,PARAMS%(1),STATUS%)
```

Chapter 5: AIO8 Driver Reference

This section provides detailed information on the functions available from the AIO8 driver. The section is divided into four sections, first is a section detailing the use of this driver, second is a task summary, third is the task reference and last is an error code summary.

Using the Driver

The Point List Concept

Most functions of this driver work with a point list. The point list is a list of point addresses in the order that you desire to have conversions performed. A point address is a number specifying the channel of the AIO8 and an AIM-16 or LVDT-8(if used). The first 16 point addresses (0-15) refer to the AIM-16 channels for an AIM-16 attached to channel 0 of the AIO8. The second 16 point addresses (16-31) refer to the 16 channels of an AIM-16 attached to channel 1 of the AIO8, and so on. Thus, with eight single ended A/D channels, a point address may be as large as 127.

If your are using LVDT-8s, then the first eight point address (0-7) refer to the LVDT-8 attached to channel 0 of the AIO8. The second eight point addresses are not used (8-15). Thus with eight single-ended A/D channels, a point address may still be as large as 127, but there would only be a maximum of 64 LVDT-8 channels.

You may install point addresses into the point list in any order, or with multiple entries for the same point address. For example the order could be 15-12-12-11-9-127-1-1-0 etc. The order that point addresses are installed in the point list is the order in which you call the driver to install them. Each new entry is appended to the end of the list.

A point list index is used by the driver to keep track of which point address is the next to be converted. After each conversion the index is incremented to the next position in the list. When the index reaches the end of the list it is automatically reset to the start of the list. If you desire to set the list index to the start of the list at any time, you may use TASK 11.

The point list is dynamic. During program operation, if you desire to clear the point list and add a different set of points, this is done quite easily using the tasks provided.

The main advantages of a point list are that conversions can be done in any order and the driver takes care of setting the AIM-16 and/or LVDT-8 channel and the AD12-8 channel, as well as gains, linearization and scaling.

Other Software Features

The driver provides the ability to use the programmable gain feature of the AIM-16. You may assign gains to a given point address directly. Each point address may have its own gain code associated with it. This is useful when differing input ranges are desired using the same AIM-16. When using this feature, the counters are not available, as they are used to set the AIM-16 gains.

The driver also provides the ability to make a function assignment to each individual point address. You may assign a thermocouple curve or a scaling range to a point address. Look up tables are contained in the driver to convert counts to the proper temperature. Reference junction compensation may also be performed.

The AIO8 combined with the AIM-16 and this driver provide an excellent tool to handle most kinds of data acquisition signals.

Task Summary

Task 0: Driver initialization.
Task 1: Check A/D operations.
Task 2: Fetch gain code for a given point address.
Task 3: Fetch point address from the point list.
Task 4: Assign gain code to a range of point addresses.
Task 5: Assign range of point addresses to the point list.
Task 6: Perform conversion of the given point address.
Task 7: Perform conversion on next point address in the point list.
Task 8: Perform multiple conversions from the point list using polling.
Task 9: Perform multiple conversions from the point list using interrupts.
Task 10: Function assignments.
Task 11: Reset operations.
Task 12: Write digital output.
Task 13: Read digital input.
Task 14: Load counter/timers.
Task 15: Read counter/timers.
Task 16: Measure frequency.
Task 17: Measure period or pulse width.

Task Reference

Task 0: Initialize

This task provides the driver with information on the card setup. This task should be called once at the beginning of the program, before any other tasks are called. If other tasks are called first, they will return error code 1.

Notes:

1. This task also calls TASK 1 to test if the card is functioning.
2. Disables all interrupt and counter activity.
3. Initializes the point list to have point addresses for each channel of the AIO8, with none for the AIM-16 or LVDT-8 (i.e. point addresses 0, 16, 32, 48 .... 112).
4. Initializes the function list for each point address to a gain code of 0 and no functions performed on conversion counts.
5. Information for the setup of the driver may be obtained either from the configuration file or from the parameters passed to the driver. The configuration file is created by using the SETUP and SETMUX setup programs, or by a word processor/text editor in the non-document mode. The driver will read the configuration file and set up the driver accordingly if params[0] = 0.
6. If no AIM-16 or LVDT-8 is to be used, use 1 for params[2].

Input:

params[0]: Base Address

params[1]: Type of initialization

- 0 = Automatic initialization using the configuration file, no other parameters are required.
- 1 = Manual initialization using information provided in the passed parameters.

params[2]: AIM-16 mode

- 0 = AIM-16 using programmable gains
- 1 = AIM-16 using manual gains.

Output:

Data: None

Error Codes:

status = 0: No error.
status = 1; Invalid task number, task > 17 or driver not initialized.
status = 2: Invalid base address, params[0] > 0x3f8 or < 0x200
status = 3: Card does not respond.
status = 15: Invalid AIM-16 mode.
status = 20: Error opening configuration file.
status = 21: Error reading configuration file.
status = 22: Invalid configuration file data.

Example:

```
inttask,params[5],status;            /* these are globally declared variables */

task = 0;
params[0] = 0x300;                   /* base address = 300 hex */
params[1] = 1;                       /* manual initialization */
params[2] = 1;                       /* AIM-16 using manual gains */
aio8drv(FP_OFF(&task),FP_OFF(params),FP_OFF(&status));   /* call the driver */
```

Task 1: Check A/D Operations

This task checks for proper operation of the analog-to-digital converter.

Notes:

1. This task is called internally by the driver when TASK 0 is called.

Input:

None

Output:

Data: None

Error Codes:

status = 0: No error.
status = 1: Invalid task number, task > 17, or driver not initialized.
status = 3: Card does not respond.

Example:

```
inttask,params[5],status;            /* these are globally declared variables */

task = 1;
aio8drv(FP_OFF(&task),FP_OFF(params),FP_OFF(&status));   /* call the driver */
```

Task 2: Fetch Gain Code for a Point Address

Returns a previously assigned gain code for a given point address.

Notes:

1. This task cannot be called if the AIM-16 is in the manual mode.

Input:

params[0]: Point address.

Output:

Data:

params[1]: Gain code for the given point address.

Error Codes:

status = 0: No error.
status = 1: Invalid task number, task > 17, or driver not initialized.
status = 5: Invalid point address, or index.
status = 18: AIM-16 in manual mode.

Example:

```
inttask,params[5],status;                    /* these are globally declared variables */

task = 2;
params[0] = 14;                              /* fetch gain code for point address 14*/
aio8drv(FP_OFF(&task),FP_OFF(params),FP_OFF(&status));   /* call the driver */
```

Task 3: Fetch Point Address for a Point List Index

Returns a previously assigned point address for a given point list index.

Notes:

None.

Input:

params[0]: Point list index.

Output:

Data:

params[1]: Point address for the given point list index.

Error Codes:

status = 0: No error.
status = 1: Invalid task number, task > 17, or driver not initialized.
status = 5: Invalid point address, or index.

Example:

```
inttask,params[5],status;          /* these are globally declared variables */

task = 3;
params[0] = 6;                     /* fetch point address for the 6th point in the point list*/
aio8drv(FP_OFF(&task),FP_OFF(params),FP_OFF(&status));   /* call the driver */
```

Task 4: Assign Gain Code to Range of Point Addresses

Assigns a given gain code to a given range of point addresses.

Notes:

1. The first point address of the range must be less than or equal to the last point address. To assign a gain code to a single point address, make the first and last point address equal.
2. The following are the possible gain codes.

GAIN Code	AIM-16 Output Range Switch Settings	
	G/2 OFF	**G/2 ON**
0	GAIN = 1	GAIN = 0.5
1	GAIN = 2	GAIN = 1
2	GAIN = 10	GAIN = 5
3	GAIN = 50	GAIN = 25
4	GAIN = 100	GAIN = 50
5	GAIN = 200	GAIN = 100
6	GAIN = 400	GAIN = 200
7	GAIN = 1000	GAIN = 500
8	AUTO RANGE	

3. These gain code settings are only meaningful if the AIM-16 is being used, and the driver has been configured for programmable gains in TASK 0.
4. A gain code of 8 indicates an auto range channel. When the point address is read, the driver will first read at a gain of 2 (gain code 1), and from this reading, determine the best gain to use for the second reading to achieve the best resolution.

Input:

params[0]: First point in point address range.
params[1]: Last point in point address range.
params[2]: Gain code to assign.

Output:

Data: None.

Error Codes:

status = 0: No error.
status = 1: Invalid task number, task > 17, or driver not initialized.
status = 5: Invalid point address, or index.
status = 6: Invalid gain code.
status = 18: AIM-16 in manual mode.

Example:

```
inttask,params[5],status;                    /* these are globally declared variables */

task = 4;
params[0] = 1;                               /* first point address in range*/
params[1] = 15;                              /* last point address in range */
params[2] = 3;                               /* gain code of 3 */
aio8drv(FP_OFF(&task),FP_OFF(params),FP_OFF(&status));   /* call the driver */
```

Task 5: Assign Point Addresses to the Point List

Assigns a range of point addresses to the point list.

Notes:

1. All point addresses added to the point list are appended to the end of the point list, after any that have been previously added, including the default point address. If you desire to start with an empty list, then use TASK 11, SUBTASK 2 to clear the point list first.
2. If the first point address is larger than the last point address, then the driver will install them in descending order.
3. Point addresses that are not on a 16 boundary (0, 16, 32 ,48 etc) are only meaningful if one or more AIM-16s or LVDT-8s are attached.

Input:

params[0]: First point address in range.
params[1]: Last point address in range.

Output:

Data: None.

Error Codes:

status = 0: No error.
status = 1: Invalid task number, task > 17, or driver not initialized.
status = 4: Point list error, list full.
status = 5: Invalid point address, or index.

Example:

```
inttask,params[5],status;                    /* these are globally declared variables */

task = 5;
params[0] = 0;                               /* first point address in range*/
params[1] = 31;                              /* last point address in range, two AIM-16s */
aio8drv(FP_OFF(&task),FP_OFF(params),FP_OFF(&status));   /* call the driver */
```

Task 6: Fetch Data from a Point Address

Perform a conversion on the point address indicated.

Notes:

1. This task does not use the point list. If you wish to fetch data from the next point in the point list then use TASK 7.
2. Point addresses that are not on a 16 boundary (0, 16, 32 ,48 etc) are only meaningful if the AIM-16 OR LVDT-8 is being used.

Input:

params[0]: Point address to fetch data from.

Output:

Data:

params[1]: Resulting conversion
params[2]: Gain code used.

Error Codes:

status = 0: No error.
status = 1: Invalid task number, task > 17, or driver not initialized.
status = 3: Card does not respond.
status = 5: Invalid point address, or index.

Example:

```
inttask,params[5],status;                /* these are globally declared variables */

task =6;
params[0] = 16;                          /* fetch data from point address */
aio8drv(FP_OFF(&task),FP_OFF(params),FP_OFF(&status));   /* call the driver */
```

Task 7: Fetch Data from next Point Address in List

Perform a conversion on the point address in the point list indicated by the point list index.

Notes:

1. Each time a point is fetched from the list, the list index is incremented. The list index can be reset to the start of the point list by using TASK 11, SUBTASK 1

Input:

None.

Output:

Data:

params[0]: Point address converted.
params[1]: Resulting conversion data.
params[2]: Gain code used.

Error Codes:

status = 0: No error.
status = 1: Invalid task number, task > 17, or driver not initialized.
status = 3: Card does not respond.
status = 5: Invalid point address, or index.

Example:

```
inttask,params[5],status;                    /* these are globally declared variables */

task = 7;
aio8drv(FP_OFF(&task),FP_OFF(params),FP_OFF(&status));   /* call the driver */
```

Task 8: Fetch Multiple Buffered Conversions

Fetch multiple conversions from the point list, using polling.

Notes:

1. Each time a point is fetched from the list, the list index is incremented. The list index can be reset to the beginning of the point list by TASK 11, SUBTASK 1.
2. This task uses two buffers, a data buffer and a point/gain buffer. Both buffers should be integer buffers of the same length. The number of conversions must not exceed the length of the shortest buffer, or else other areas of computer memory may be corrupted, causing unpredictable computer behavior. The driver has no criteria to evaluate the validity of the pointer. It is incumbent upon the application program to supply a valid buffer pointer.
3. The point and gain for each analog input is returned in the point/gain buffer. The point address and gain are packed into one integer with the point address in the upper eight bits and the gain in the lower eight bits.
4. The buffers must be declared globally or the driver will not be able to find their segment.

Input:

params[0]: Offset of the data buffer address.
params[1]: Offset of the point/gain buffer address.
params[2]: Number of conversions to make.

Output:

Data:

params[3]: Number of conversions completed.
The buffers will contain the conversions and the point/gain data respectively.

Error Codes:

status = 0: No error.
status = 1: Invalid task number, task > 17, or driver not initialized.
status = 3: Card does not respond.
status = 5: Point list error, list empty.

Example:

```
inttask,params[5],status;               /* these are globally declared variables */
intdatbuf[100],chnbuf[100];             /* these are globally declared variables */

task = 8;
params[0] = FP_OFF(datbuf);             /* pass offset of data buffer */
params[1] = FP_OFF(chnbuf);             /* pass offset of point/gain  buffer */
params[2] = 100;                        /* number of conversions */
aio8drv(FP_OFF(&task),FP_OFF(params),FP_OFF(&status));   /* call the driver */
```

Task 9: Interrupt Driven Data Acquisition

Provides subtasks to perform buffered data acquisition using interrupts. Sub task functions include initiating the interrupt conversions, checking for completion and stopping the interrupt process.

Notes:

1. Each time a point is fetched from the list, the list index is incremented. The list index can be reset to the beginning of the point list by TASK 11.
2. This task uses two buffers, a data buffer and a point/gain buffer. Both buffers should be integer buffers of the same length. The number of conversions must not exceed the length of the shortest buffer., or else other areas of computer memory may be corrupted, causing unpredictable computer behavior. The driver has no criteria to evaluate the validity of the pointer. It is incumbent upon the application program to supply a valid buffer pointer.
3. The point and gain for each analog input is returned in the point/gain buffer. The point address and gain are packed into one integer with the point address in the upper eight bits and the gain in the lower eight bits.
4. The buffers must be declared globally or the driver will not be able to find their segment.
5. This task has several functions, each having its own required parameters.
6. If the timers are used to generate the start-conversion signals, then they should be configured using TASK 14, before calling TASK 9.
7. SUBTASK 3 is used to disable interrupts before completion of the scan. When the scan completes normally, the interrupts are disabled automatically.

8. Interrupts are generated by an external source. This source is connected to the card via pin 24 of the external connector. One of the on board counters may be used for this source, providing that any AIM-16's in the system are being operated in the manual mode. If using a counter you must set up the counter using TASK 14, then connecting the counter's output to pin 24.

Input:

params[0]: Subtask to perform, 1, 2, or 3.

1. Initiate interrupt data acquisition.
   - params[1]: Interrupt level (IRQ)
   - params[2]: Number of conversion to make.
   - params[3]: Offset of the data buffer address.
   - params[4]: Offset of the point/gain buffer address.

2: Check for end of interrupt scan.

3: Disable the interrupt operation.

Output:

Data:

SUBTASK 1: The buffers will contain the conversions and the point/gain data respectively.

SUBTASK 2: params[1] = 0 if scan complete, task number if still in progress.

Error Codes:

status = 0: No error.
status = 1: Invalid task number, task > 17, or driver not initialized.
status = 3: Card does not respond.
status = 5: Point error, point list is empty.
status = 7: Invalid number of conversions, not between 1 and 32767.
status = 10: Interrupt task already active.
status = 11: Interrupt not between 2 and 7.
status = 12: Interrupt already unassigned. (SUBTASK 3)
status = 13: Invalid subtask, not 1, 2 or 3.
status = 14: Invalid trigger mode, not 1 or 2.

Example:

```
inttask,params[5],status;              /* these are globally declared variables */
intdatbuf[100],chnbuf[100];            /* these are globally declared variables */

task = 9;
params[0] = 1;                         /* initiate interrupt scan */
params[1] = 5;                         /* use IRQ5 */
params[2] = 100;                       /* do 100 conversions on this scan */
params[3] = FP_OFF(datbuf);            /* pass offset of data buffer */
params[4] = FP_OFF(chnbuf);            /* pass offset of point/gain  buffer */
aio8drv(FP_OFF(&task),FP_OFF(params),FP_OFF(&status));   /* call the driver */
params[0] = 2;                         /* check for end of scan process */
do
  {
aio8drv(FP_OFF(&task),FP_OFF(params),FP_OFF(&status));   /* call the driver */
}
while (params[1] != 0);                 /* wait until end of scan
                                       /* or if you do not want to wait until end of scan */
params[0] = 3;                         /* stop interrupt process sub task */
aio8drv(FP_OFF(&task),FP_OFF(params),FP_OFF(&status));   /* call the driver */
```

Task 10: Thermocouple/Function Assignment

Provides subtasks to assign thermocouple curves and scaling factors to a given point address. A subtask is also provided to use the thermocouple tables to linearize values passed to it.

Notes:

1. The built-in NBS tables are designed to convert A/D counts to temperature directly. When using SUBTASK 1, the driver expects the passed counts to be multiplied by 16 if using the bipolar mode, and multiplied by 8 if using a unipolar mode.
2. Curves are assigned to a point address by calling SUBTASK 2 with the ASCII code of one of the curves listed in the table that follows. Also, temperature units are assigned in the same manner.
3. If reference junction compensation using the AIM-16 on board sensor is desired, then assign this sensor to the point list as the first channel of a given AIM-16 (ie. 0, 16,32,48 etc). Make sure that the TMP jumpers are installed on the AIM-16. Finally, assign the curve "T" to this point address using SUBTASK 2. Any other point addresses on the AIM-16 will now be junction compensated automatically by the driver each time a point address is converted.
4. The reference junction circuit on the AIM-16 card generates 24.4 mV/oC. The counts read in at a gain of 1 are 2.44 millivolts/count. Thus, each count represents 0.1o C.

5. When thermocouple curves are assigned to a point address, it is also required to set that point address to a particular gain using TASK 4. These gains are presented in the following table. Note that two gain codes are presented for each thermocouple type, the one you use will depend on the setting of the G/2 switch on the AIM-16. If G/2 is OFF, use the lower gain code, if G/2 is ON then use the higher gain code.

T/C TYPE	GAIN	GAIN CODE	µVOLTS/COUNT
b	200	5/6	12.207
e	50	3/4	48.828
j	100	4/5	24.414
k	50	3/4	48.828
r	200	5/6	12.207
s	200	5/6	12.207
t	200	5/6	12.207
RTD TYPE	**GAIN**	**GAIN CODE**	**µVOLTS/COUNT**
a	100	4/5	24.414
u	100	4/5	24.414

6. Temperature is returned in increments of 1/10th degree. For example, 100 degrees would be returned as 1000.
7. SUBTASK 3 can be used to force the driver to return values in units determined by the user rather than counts. For example, you might desire values returned in millivolts. In such a case, assuming the bipolar mode, scale factors of -5000 and +5000 would be passed in the call to SUBTASK 3.
8. For platinum RTD's, there are two curves; "a" for sensors with 392 alpha and "u" for sensors with 385 alpha.
9. TASK 10 does not initiate any conversions, but sets up functions that will be performed automatically whenever conversion are done using tasks 6, 7, 8, 9 or 16.

Input:

params[0]: Subtask to perform, 1, 2, 3 or 4.

- 1: Perform linearization of the given data.
  - params[1]: ASCII code for lower case letter of curve, or upper case T for reference junction.
  - params[2]: counts (see note 1)
- 2: Assign thermocouple curve to a point address.
  - params[1]: point address
  - params[2]: ASCII code for lower case letter of curve, or upper case T for reference junction.
  - params[3]: ASCII code for upper case letter of the desired temperature units, C or F.

3: Assign scaling factor to a point address.
   params[1]: point address
   params[2]: Lower scaling term.
   params[3]: Upper scaling term.
4: Replicate a point address function assignment to a range of point addresses.
   params[1]: source address to replicate
   params[2]: first point address in destination range
   params[3]: last point address in destination range.

Output:

Data:

SUBTASK 1:

params[3]:temperature in tenths of OF.
params[4]:temperature in tenths of OC.

Error Codes:

status = 0: No error.
status = 1: Invalid task number, task > 20, or driver not initialized.
status = 5: Point error, point address out of range.
status = 13: Invalid subtask, not between 1 and 4.
status = 16: Invalid curve.

Example:

```
inttask,params[5],status;                    /* these are globally declared variables */

task = 10;
                                             /*  linearize the passed value */
params[0] = 1;                               /* manual linearization subtask */
params[1] = 't';                             /* for t type thermocouple */
params[2] = 1801;                            /* counts * 16 at gain of 200 */
aio8drv(FP_OFF(&task),FP_OFF(params),FP_OFF(&status));   /* call the driver */
                                             /* values returned in params[3] and params[4] */
                                             /* assign curve to a point address */
params[0] = 2;                               /* curve assignment  subtask */
params[1] = 0;                               /* first point address on first AIM-16 */
params[2] = 'T';                             /* T for reference junction */
params[3] = 'F';                             /* F for degrees F */
aio8drv(FP_OFF(&task),FP_OFF(params),FP_OFF(&status));   /* call the driver */
                              /* assign a range of ±5 volts in millivolt increments to a point address.
params[0] = 3;                               /* range assignment subtask */
params[1] = 22;                              /* point address to assign */
params[2] = -5000;                           /* lower range */
params[3] = 5000;                            /* upper range */
aio8drv(FP_OFF(&task),FP_OFF(params),FP_OFF(&status));   /* call the driver */
```

```
                    /* replicate the assignment for point address 22 to point addresses 23-40 */
params[0] = 4;                          /* replication  subtask */
params[1] = 22;                         /* source point address to replicate */
params[2] = 23;                         /* lower point address in destination range */
params[3] = 40;                         /* upper point address in destination range */
aio8drv(FP_OFF(&task),FP_OFF(params),FP_OFF(&status));   /* call the driver */
```

Task 11: Reset Functions

Performs various reset functions on the point list and function/curve assignment tables. Also provides a subtask to allow for the sample and hold settle time, in higher speed computers.

Notes:

1. SUBTASK 5 is provided to set the sample and hold settle time. High speed 80286 and 80386 computers often will start a conversion before the sample and hold has had time to settle after changing a channel on the AIM-16. A value of 25-50 for the delay loop is usually sufficient for an 80386 machine.

Input:

params[0]:   Subtask to perform, 1, 2, 3, 4 or 5.

- 1: Reset the point list index to first point address in the point list.
- 2: Clears all point addresses from the point list.
- 3: Resets the point list to the default conditions, as described in TASK 0.
- 4: Clears all curve and scaling assignments.
- 5: Set the sample and hold settle time.
  - params[1]:settle time count

Output:

Data: None.

Error Codes:

status = 0:   No error.
status = 1:   Invalid task number, task > 17, or driver not initialized.
status = 15:   Invalid reset sub task, not between 0 and 5.

Example:

```
inttask,params[5],status;               /* these are globally declared variables */

task = 11;
params[0] = 5;                          /* set settle time  sub task */
params[1] = 50;                         /* settle time count of 50 */
aio8drv(FP_OFF(&task),FP_OFF(params),FP_OFF(&status));/* call the driver */
```

Task 12: Digital Output

Writes to the digital output bits.

Notes:

1. Output values are not checked for proper range. If a value greater than fifteen is sent, the driver will only output fifteen, which is the lower four bits.
2. If an AIM-16 is being used, this task should not be called, as these digital output bits are used to set the channel on the AIM-16.

Input:

params[0]: Value to output, 0 to 15 decimal.

Output:

Data: None.

Error Codes:

status = 0:No error.
status = 1:Invalid task number, task > 17, or driver not initialized.

Example:

```
inttask,params[5],status;                /* these are globally declared variables */

task = 12;
params[0] = 15;                          /* set first  4 output bits high */
aio8drv(FP_OFF(&task),FP_OFF(params),FP_OFF(&status));   /* call the driver */
```

Task 13: Digital Input

Reads the digital input bits.

Notes:

1. Returns the state of the three digital input bits.

Input:

None.

Output:

Data:

params[1]: Digital input value.

Error Codes:

status = 0:No error.
status = 1:Invalid task number, task > 17, or driver not initialized.

Example:

```
inttask,params[5],status;          /* these are globally declared variables */

task = 13;
aio8drv(FP_OFF(&task),FP_OFF(params),FP_OFF(&status));   /* call the driver */
```

Task 14: Counter/Timer Setup

Load the given counter/timer with a count value and mode.

Notes:

1. For a complete discussion of the counter/timers, see Programmable Interval Timer.
2. If the AIM-16 is in the programmable gain mode, this task cannot be called. The driver will return error code 18 if you call this task, after having called TASK 0 with the AIM-16 mode set to programmable.

Input:

params[0]: counter number 0, 1 or 2.
params[1]: counter mode, between 0 and 5.
params[2]: counter load count.

Output:

Data: None.

Error Codes:

status = 0:No error.
status = 1:Invalid task number, task > 17, or driver not initialized.
status = 8:Invalid counter, not 0, 1 or 2.
status = 9:Invalid counter mode, not between 0 and 5.
status = 18:AIM-16 programmable mode set.

Example:

```
inttask,params[5],status;          /* these are globally declared variables */

task = 14;
params[0] = 1;                     /* counter 1 */
params[1] = 3;                     /* counter mode 3, square wave generator */
params[2] = 100;                   /* counter load value, acts as divide by 100 */
aio8drv(FP_OFF(&task),FP_OFF(params),FP_OFF(&status));   /* call the driver */
```

Task 15: Read Counter/Timer Count

Reads the count of the given counter/timer.

Notes:

1. For a complete discussion of the counter/timers, see Programmable Interval Timer.
2. Counter/timer is latched before read.
3. If the AIM-16 is in the programmable gain mode, this task cannot be called. The driver will return error code 18 if you call this task, after having called TASK 0 with the AIM-16 mode set to programmable.

Input:

params[0]: counter number 0, 1 or 2.

Output:

Data:

params[1]:counter/timer count.

Error Codes:

status = 0:No error.
status = 1:Invalid task number, task > 17, or driver not initialized.
status = 8:Invalid counter, not 0, 1 or 2.

Example:

```
inttask,params[5],status;                    /* these are globally declared variables */

task = 15;
params[0] = 1;                               /* counter 1 */
aio8drv(FP_OFF(&task),FP_OFF(params),FP_OFF(&status));   /* call the driver */
```

Task 16: Measure Frequency

Measures an unknown frequency.

Notes:

1. All calculations performed by the driver, as well as quantities presented in this task assume a 4.77MHz bus clock. If you have a different clock speed, try calibrating the task with a known frequency until you have discovered the proper value to pass and the conversion value.
2. If the AIM-16 is in the programmable gain mode, this task cannot be called. The driver will return error code 18 if you call this task, after having called TASK 0 with the AIM-16 mode set to programmable.

3. Counter 2 is programmed to output at a 1mS pulse rate. The load value is 2385, which gives a 1mS pulse rate for a 4.77 MHZ bus clock. To determine the output value for your computer, divide 4.77Mhz by your bus clock to find the pulse rate. The output of counter 2 should be connected to the input of counter 1 by connecting pin 4 to pin 6 on the external I/O connector.
4. he output of counter 1 is connected to the gate of counter 0 and to IP2 by connecting pins 5, 21 and 26 together. The unknown frequency is connected to counter 0's clock by connecting it between pin 2 and common(pin 11 or 28). The value passed in params[0] is loaded into counter 1 and provides a multiple of the pulse rate calculated in 3).
5. The return value from the task is the number of counts during the time interval from 4). The frequency is computed using the following equation:

$$\text{frequency} = \frac{\text{params[1]} * 1000}{\text{params[0]} * \text{pulse rate (in mS)}}$$

6. Accuracy of 0.1% is achieved with this task. For better accuracy with lower frequencies, providing the waveform is symmetrical, use TASK 17.

Input:

params[0]: pulse rate mulitples

Output:

Data:

params[1]: number of counts during the time from 4).

Error Codes:

status = 0: No error.
status = 1: Invalid task number, task > 17, or driver not initialized.
status = 18: AIM-16 in programmable gain mode.

Example:

```
inttask,params[7],status;                /* these are globally declared variables */
doublefrequency;

task = 16;
params[0] = 100                          /* 100 ms pulse rate, 4.77 MHz bus clock */
aio8drv(FP_OFF(&task),FP_OFF(params),FP_OFF(&status));   /* call the driver */
frequency = (double) (params[1] * 1000)/params[0];        /* frequency calculation */
```

Task 17: Measure Pulse Width

Measure an unknown pulse width with counter 2.

Notes:

1. If the AIM-16 is in the programmable gain mode, this task cannot be called. The driver will return error code 18 if you call this task, after having called TASK 0 with the AIM-16 mode set to programmable.
2. The unknown signal should be connected to pin 23 and pin 26.
3. The maximum pulse width that can be measured is 65.54mS.
4. Multiply the result returned in params[0] by the reciprocal of your bus clock frequency divided by 2 to get the actual pulse width.

Input:

None

Output:

Data:

params[0]: Change in counts.

Error Codes:

status = 0: No error.
status = 1: Invalid task number, task > 17, or driver not initialized.
status = 18: AIM-16 is in programmable gain mode.

Example:

```
inttask,params[5],status;                 /* these are globally declared variables */
double
task = 17;
aio8drv(FP_OFF(&task),FP_OFF(params),FP_OFF(&status));   /* call the driver */
width = (double) params[0] * 0.41905;     /* calculation based on 4.77 MHZ bus clock */
```

Summary of Error Codes

1: Invalid task number: The task number does not fall within the range of 0 through 17. This error code also occurs if any task is selected before a successful initialization with TASK 0.

2: Invalid base address: The base I/O address does not fall within the range of 100 hex through 3F8 hex.

3: A/D failed: The EOC (end-of-conversion) signal did not change state. This is usually because the base address has not been set properly.

4: Point list is full: The point list can only hold 128 entries.

5: Invalid point address: The point address does not fall within the range of 0 through 127.

6: Invalid gain code: The gain code does not fall within the range of 0 through 8.

7: Invalid TASK 11 subtask: The subtask does not fall within the range of 0 through 5.

8: Invalid counter/timer number: The counter/timer number is not 0, 1 or 2.

9: Invalid mode: The counter/timer mode does not fall within the range of 0 through 5.

10: Interrupt process already set: The interrupt process can only be set up once. A subsequent request has been made to set up the interrupt process without previously resetting the mode.

11: Invalid interrupt number: The interrupt number does not fall within the range of 2 through 7.

12: No interrupt process is set: A request has been made to reset an interrupt process that has not previously been set.

13: Invalid SUBTASK number: SUBTASK specified is outside the valid range for a given task.

14: Invalid data buffer: Data buffer is not valid for interrupt driven data acquisition.

15: Invalid AIM-16 mode: AIM-16 mode should be 0(programmable gains) or 1(manual gains).

16: Invalid curve specified: The curve code letter specified is not a valid code.

17: Not used.

18: An attempt to use a task that requires the AIM-16 to be in programmable gain mode has been called when the AIM-16 was set for manual gain mode.

19: An attempt to use a task that requires the AIM-16 to be in manual gain mode has been called when the AIM-16 was set for programmable gain mode.

20: Error opening configuration file.

21: Error reading configuration file.

22: Configuration file data are not correct.

Chapter 6: A/D Converter Applications

Connecting Analog Inputs

The AIO8 provides eight channels of single-ended input. Single-ended configuration means that you have only one input relative to ground. A differential input provides two inputs and the signal corresponds to the voltage difference between these two inputs. The single-ended configuration is suitable only for "floating" sources; i.e., a signal source that does not have any connection to ground at the source. To use differential connections to the AIO8, the AIM-16 multiplexer card must be added. The AIM-16 supplies 16 channels of differential input.

Thus, if the signal source has one side connected to a local ground, the AIO8/AIM-16 combinations should be used. A differential input responds only to difference signals between the high and low inputs. In practice, the signal source ground will not be at exactly the same voltage as the computer ground where the AIO8/AIM-16 combination is because the two grounds are connected through ground returns of the equipment and the building wiring. The difference between the ground voltages forms a common mode voltage (i.e., a voltage common to both inputs) that a differential input rejects up to a certain limit. In the case of the AIO8/AIM-16 combination, the common mode voltage limit is ±10V.

It's important to understand the difference between input types, how to use them effectively, and how to avoid ground loops. Misuse of inputs is the most common difficulty that users experience in applying and obtaining the best performance from data acquisition systems.

Comments on Noise Interference

Noise is generally introduced into analog measurements from two sources: (a) ground loops and (b) external noise. In both cases, use of good wiring practice will reduce and sometimes eliminate the noise. A key point with regard to ground or return wiring is that in an analog/digital "system", digital circuits should have a separate ground system from analog circuits with only a single common point. The reason for separate ground busses is that digital circuits, by their very nature, generate considerable high frequency noise as they rapidly change state.

Ground loops occur when AC noise and DC offset are added in series with a grounded signal source if the source ground is at a different potential than the A/D's analog ground. If there is an ohmic resistance between the source ground and the A/D's ground, the resultant current flow causes a voltage to be developed and a "ground loop" exists. If the signal is measured in a single-ended mode, that voltage is added to the source signal thereby creating an error. The best way to avoid ground loop errors is to use good wiring practice as described above. If this is not possible, use of a differential measurement mode will minimize errors.

Input Range and Resolution Specifications

Resolution of an A/D converter is usually specified in number of bits; i.e. 8 bits, 12 bits, etc. Input range is specified in volts; i.e. 0-5V, ±10V, ±20mV, etc. To determine the voltage resolution of an A/D converter, simply divide the full scale voltage range by the number of parts of resolution. For example, for a bipolar range of ±5V, a 12-bit A/D resolves the input into 4096 parts. Thus, voltage resolution (the "weight" of one bit) is 2.44mV.

If an amplifier is incorporated in the circuit providing gain, then divide the voltage resolution by the gain of the amplifier, then divide by 4096. For example, a 12-bit A/D with ±5V full-scale input range and an amplifier gain of 100 will provide an overall input resolution of about 24.4µV.

Current Measurements

Current signals can be converted to voltage for measurement by the A/D converter by addition of a shunt resistor installed across the input terminals. For example, to accommodate 4-20mA current transmitter inputs, connect a 250Ω shunt resistor across the A/D input terminals. The resultant 1-5V signal can then be measured. The screw terminal accessory board, for example, includes a breadboard area with plated through holes that allow insertion of shunt resistors.

If an AIM-16 multiplexer expansion card is being used, pre-wired pads are provided on the AIM-16. If all the inputs are 4-20mA range current inputs from current transmitters, then there is a configuration of the multiplexer expansion board called AIM-16I. That model includes the shunt resistors and has offset and gain set such that the "live zero" is compensated for and the full 12-bit resolution of the A/D is realized.

Note

Accuracy of measurement will be directly affected by the accuracy of these resistors. Accordingly, precision resistors should be used. Also, if the ambient temperature will vary significantly, these precision resistors should be low-temperature-coefficient wire-wound resistors.

Measuring Large Voltages

Voltages larger than the input range of the A/D can be measured by using a voltage divider. As above, it is necessary to use precision resistors. Also if the raw voltage is a direct analog of a parameter being measured, then it will be necessary to apply a scale factor in software in order to arrive at the correct engineering units.

Adding More Analog Inputs

You can add sub-multiplexers to any or all of the analog inputs of AIO8. The AIM-16 provides capability for 16 channels per input plus a common instrumentation amplifier. Up to eight AIM-16's can be added to one AIO8 providing a total input capability up to 128 channels.

Precautions - Noise, Ground Loops, and Overloads

Unavoidably, data acquisition applications involve connecting external things to the computer. DO NOT get inputs mixed up with the AC line. An inadvertent short can instantly cause extensive damage. As an aid to avoid this problem:

a. Avoid direct connection to the AC line.

b. Make sure that all connections are secure so that signal wires are not likely to come loose and short to high voltages.

c. Use isolation amplifiers and transformers where necessary. There are two types of ground connections on the rear connector of AIO8. These are called Power Ground and Low Level Ground. Power ground is the noisy or dirty ground that is meant to carry all digital signals and heavy (power supply) currents. Low Level Ground is the signal ground for all analog input functions. It is only meant to carry signal currents (less than a few milliamperes) and is the ground reference for the A/D converter. Due to connector contact resistance and cable resistance there may be many millivolts difference between the two grounds even though they are connected together and to the computer and power line grounds on the AIO8 card.

Chapter 7: Programmable Interval Timer

The AIO8 contains a type 8254 programmable counter/timer which allows you to implement such functions as a Real-Time Clock, Event Counter, Digital One-Shot, Programmable Rate Generator, Square-Wave Generator, Binary Rate Multiplier, Complex Wave Generator, and/or a Motor Controller. The 8254 is a flexible but powerful device that consists of three independent, 16-bit, presettable, down counters. Each counter can be programmed to any count between 1 or 2 and 65,535 in binary format, depending on the mode chosen.

On the AIO8 these three counters are designated Counter #0, Counter #1, and Counter #2. Counter #0 and counter #1 have the gate, output and clock connections fully accessible via the I/O connector. Counter #2 receives clock inputs from a ½ multiple of the PC bus clock. The output and gate of counter #2 is also available at the I/O connector. If the AIM-16 is being used with programmable gain, then all counters are required for setting the gains on the AIM-16.

Operational Modes

The 8254 modes of operation are described in the following paragraphs to familiarize you with the versatility and power of this device. For those interested in more detailed information, a full description of the 8254 programmable interval timer can be found in the Intel (or equivalent manufacturers) data sheets. The following conventions apply for use in describing operation of the 8254:

Clock:	A positive pulse into the counter's clock input.
Trigger:	A rising edge input to the counter's gate input.
Counter Loading:	Programming of a binary count into the counter.

Mode 0: Pulse on Terminal Count

After the counter is loaded, the output is set low and will remain low until the counter decrements to zero. The output then goes high and remains high until a new count is loaded into the counter. A trigger enables the counter to start decrementing. This mode is commonly used for event counting with Counter #0.

Mode 1: Retriggerable One-Shot

The output goes low on the clock pulse following a trigger to begin the one-shot pulse and goes high when the counter reaches zero. Additional triggers result in reloading the count and starting the cycle over. If a trigger occurs before the counter decrements to zero, a new count is loaded. Thus, this forms a re-triggerable one-shot. In mode 1, a low output pulse is provided with a period equal to the counter count-down time.

Mode 2: Rate Generator

This mode provides a divide-by-N capability where N is the count loaded into the counter. When triggered, the counter output goes low for one clock period after N counts, reloads the initial count, and the cycle starts over. This mode is periodic, the same sequence is repeated indefinitely until the gate input is brought low. This mode also works well as an alternative to mode 0 for event counting.

Mode 3: Square Wave Generator

This mode operates periodically like mode 2. The output is high for half of the count and low for the other half. If the count is even, then the output is a symmetrical square wave. If the count is odd, then the output is high for (N+1)/2 counts and low for (N-1)/2 counts. Periodic triggering or frequency synthesis are two possible applications for this mode. Note that in this mode, to achieve the square wave, the counter decrements by two for the total loaded count, then reloads and decrements by two for the second part of the wave form.

Mode 4: Software Triggered Strobe

This mode sets the output high and, when the count is loaded, the counter begins to count down. When the counter reaches zero, the output will go low for one input period. The counter must be reloaded to repeat the cycle. A low gate input will inhibit the counter. This mode can be used to provide a delayed software trigger for initiating A/D conversions.

Mode 5: Hardware Triggered Strobe

In this mode, the counter will start counting after the rising edge of the trigger input and will go low for one clock period when the terminal count is reached. The counter is retriggerable. The output will not go low until the full count after the rising edge of the trigger.

Programming

On the AIO8, the 8254 counters occupy the following addresses:

BASE ADDRESS + 4: Read/Write Counter #0
BASE ADDRESS + 5: Read/Write Counter #1
BASE ADDRESS + 6: Read/Write Counter #2
BASE ADDRESS + 7: Write to Counter Control register

The counters are programmed by writing a control byte into a counter control register at BASE ADDRESS + 7. The control byte specifies the counter to be programmed, the counter mode, the type of read/write operation, and the modulus. The control byte format is as follows:

B7	B6	B5	B4	B3	B2	B1	B0
SC1	SC0	RW1	RW0	M2	M1	M0	BCD

SC0-SC1: These bits select the counter that the control byte is destined for.

SC1	SC0	Function
0	0	Program Counter 0
0	1	Program Counter 1
1	0	Program Counter 2
1	1	Read back Command

RW0-RW1: These bits select the read/write mode of the selected counter.

RW1	RW0	Counter Read/Write Function
0	0	Counter Latch Command
0	1	Read/Write LS Byte
1	0	Read/Write MS Byte
1	1	Read/Write LS Byte, then MS Byte

M0-M2: These bits set the operational mode of the selected counter.

Mode	M2	M1	M0
0	0	0	0
1	0	0	1
2	X	1	0
3	X	1	1
4	1	0	0
5	1	0	1

BCD: Set the selected counter to count in binary (BCD bit = 0) or BCD (BCD bit = 1).

Reading and Loading the Counters

If you attempt to read an active counter, you will most likely get erroneous data. This is partly caused by carries rippling through the counter during the read operation. Also, the low and high bytes are read sequentially rather than simultaneously and, thus, it is possible that carries will be propagated from the low to the high byte during the read cycle. To circumvent these problems, you should perform a counter-latch operation in advance of the read cycle. To do this, load the RW1 and RW2 bits with zeroes. This instantly latches the count of the selected counter(selected via the SC1 and SC0 bits) in a 16-bit hold register. A subsequent read operation on the selected counter returns the held value. Latching is the best way to read an active counter without disturbing the counting process. You can only rely on directly read counter data if the counting process is suspended while reading, by bringing the gate low, or by halting the input pulses.

For each counter you must specify in advance the type of read or write operation that you intend to perform. You have a choice of loading/reading (a) the high byte of the count, or (b) the low byte of the count, or (c) the low byte followed by the high byte.

Programming Examples

Using Counter #0 as a Pulse Counter

Note that the counters are "down" counters so, when resetting them, it's better to load them with a full count value of 65,535 rather than zero.

```
outportb(BASEADDRESS + 7,0x30);      /* counter 0, mode 0 */
outportb(BASEADDRESS + 4,0xff);      /* counter 0 low load byte */
outportb(BASEADDRESS + 4,0xff);      /* counter 0 high load byte */
```

Reading Counter #0

```
outportb(BASEADDRESS + 7,0);         /* counter 0, latch command */

/* read in both bytes of the latched value and combine into an integer */
value = inportb(BASEADDRESS + 4) +  (inportb(BASEADDRESS + 4) * 256;
```

Programming Examples Using the AIO8DRV Driver

In practice, TASKS 14 and 15 of the AIO8DRV driver can be used to perform equivalent operations to the above examples with fewer programming steps.

For counting pulses, the counter configuration is not of great importance because you will only be using the countdown capabilities of the counter. Mode 2 is as good as any other choice for pulse counting. As in the previous example, load Counter #0 with a full scale count of 65,535 (hex FFFF) using TASK 14 of the driver. While loading the counter, counting can be inhibited by holding the gate input, pin 21, low.

```
task = 14;                 /* counter setup mode task */
params[0] = 0;             /* setup counter #0 */
params[1] = 2;             /* set counter #0 mode to 2 */
params[2] = 0xffff;        /* set counter #0 count to ffff hex (65535) */
aio8drv(FP_OFF(task),FP_OFF(params),FP_OFF(status));   /* call the driver */
```

Next, apply the number of pulses to be counted. The gate input, pin 21, must now be high or can be taken high for some fixed time interval to control the number of pulses counted. You can read the new count using TASK 15 of the driver:

```
task = 15;                 /* read counter count task */
params[0] = 0;             /* read counter #0 */
aio8drv(FP_OFF(task),FP_OFF(params),FP_OFF(status));   /* call the driver */
```

Upon return, params[1] contains the counter contents.

Generating Square Waves of Programmed Frequency

Frequency of output is a direct function of the frequency of the clock input and of the count loaded into the counter. The minimum count (or divisor) is 2 and the maximum is 65535.

Calculating what divisor to use for a specific output frequency is straightforward. If, for example, you desire a 1KHz output and your clock is 5MHz, divide by 1000 and find that the count to be loaded into counter #0 should by 5000.

Measuring Frequency and Period

TASK 16 of the driver describes measuring frequency and TASK 17 describes measuring pulse width.

Generating Time Delays

There are four methods of using counter #0 or counter #1 to generate programmable time delays.

Pulse on Terminal Count

After loading, the counter output goes low. Counting is enabled when the gate goes high. The counter output will remain low until the count reaches zero, at which time the counter output goes high. The output will remain high until the counter is reloaded by a programmed command. If the gate goes low during countdown, counting will be disabled as long as the gate input is low.

Programmable One-Shot

The counter need only be loaded once. The time delay is initiated when the gate input goes high. At this point the counter output goes low. If the gate input goes low, counting continues but a new cycle will be initiated if the gate input goes high again before the timeout delay has expired; i.e., is re-triggerable. At the end of the timeout, the counter reaches zero and the counter output goes high. That output will remain high until re-triggered by the gate input.

Software Triggered Strobe

This is similar to Pulse-on-Terminal-Count except that, after loading, the output goes high and only goes low for one clock period upon timeout. Thus, a negative strobe pulse is generated a programmed duration after the counter is loaded.

Hardware Triggered Strobe

This is similar to Programmable-One-Shot except that when the counter is triggered by the gate going high, the counter output immediately goes high, then goes low for one clock period at timeout, producing a negative-going strobe pulse. The timeout is re-triggerable; i.e., a new cycle will commence if the gate goes high before a current cycle has timed out.

Appendix A: Linearization

A common requirement encountered in data acquisition is to linearize or compensate the output of non-linear transducers such as thermocouples, flowmeters, etc. The starting point for any linearizing algorithm is a knowledge of the calibration curve (input/output behavior) of the transducer. This may be derived experimentally or may be available in manufacturer's data or standard tables.
There are several approaches to linearization. The two most common are piecewise linearization using look-up tables, and the use of a mathematical function to approximate the non-linearity. Amongst the mathematical methods, polynomial expansion is one of the easiest to implement. The utility program, POLY.EXE, allows you to generate up to a 10th order polynomial approximation. For most practical applications, a fifth-order polynomial approximation is usually adequate.
Before you start the program have the desired input/output data or calibration data handy. This will be in the form of x and f(x) values where x is the input to your system and f(x) is the resulting output.

To run the program, type POLY and ENTER at the command line. The program will then prompt you for the desired order of the polynomial, then the number of pairs that you wish to use to generate the polynomial. You then enter the data pairs and the polynomial is computed and displayed.

For example, given the following data points, let's generate a 5th order polynomial to approximate this function:

x	0	1	2	3	4	5	6	7	8	9	10
f(X)	3	2	3	5	3	4	3	2	2	3	2

The order of the polynomial that you desire will be 5 and the number of data points that you enter will be 11. After the data points are entered, the program gives the following output:

For the polynomial:

f(x) = C(0) + C(1)x1 + C(2)x2 + C(3)x3 + C(4)x4 + C(5)x5

The coefficients will be:

COEFFICIENT (5) : -0.003151
COEFFICIENT (4) : 0.081942
COEFFICIENT (3) : -0.740668
COEFFICIENT (2) : 2.635998
COEFFICIENT (1) : -2.816607
COEFFICIENT (0) : 2.956044

QUALITY OF SOLUTION (sum of the errors squared): 2.797989

The goal is to make the quality as close to 0 as possible.

Note the quality of solution. The program checks the resulting polynomial with the data pairs that you entered. It computes the f(x) values for each x value entered using the polynomial, subtracts the result from the supplied value of f(x), and then squares the result. The squared results are then summed to compute the QUALITY. If the computed f(x) values were exact, this value would be 0. But, since this is an approximation, this value will usually be something greater than 0.

The QUALITY can be used to indicate how good a particular solution is. If the range of points is very wide or if the points make transition from negative to positive values, then QUALITY will suffer accordingly. For these cases, it may be better to use multiple polynomials rather than just one.

As an example, the following data are taken from the NIST tables for type T thermocouples:

x	-6.258	-5.603	-4.468	-3.378	-1.182	0	2.035
f(x)	-270	-200	-150	-100	-50	0	50

4.277	6.702	9.286	12.01	14.86	17.82	20.87
100	150	200	250	300	350	400

If we take all the data and compute one 5th order polynomial, the QUALITY is 473.543732; not very good. Now divide the data into two polynomials; one on the negative side including 0 and one on the positive side also using 0. The results will show a QUALITY of 90.732620 for the negative side and a QUALITY of 0.005131 for the positive side. Thus, by using two polynomials, you have made the positive side very accurate and dramatically improved the negative side.

Accuracy of the negative side can be further improved by adding points. For example, add the following pairs to the negative side of the polynomial for a type T thermocouple:

x	-6.181	-5.167	-4.051	-2.633
f(x)	-250	-175	-125	-75

If you run the new data, the QUALITY is improved to 69.555611, but still perhaps not as good as you would like.

Thus, you may use the QUALITY as a means to determine how good the polynomial is. You can experiment with both order and number of data points until you are satisfied with the solution. Incidentally, this example also shows that the smaller the range of x values, the better the solution.

The computational method used is a least squares solution using Gauss Elimination with partial pivoting to improve accuracy.

Appendix B: Cabling and Connector Information

AIO8 Output Connector Pin Assignments

Connections are made to the AIO8 card via a 37-pin D type connector that extends through the back of the computer case. The female mating connector can be a Cannon #DC-37S for soldered connections or insulation displacement flat cable types such as AMP #745242-1 may be used. The wiring may be directly from the signal sources or may be on ribbon cable from screw terminal accessories.   The pin assignments are as follows:

Pin	Name	Function
1	+12VDC	+12VDC Power from the Computer Bus
2	CTR0 CLK	Counter 0 Clock
3	CTR0 OUT	Counter 0 Output, programmable gain control for AIM-16 (LSB)
4	CTR1 CLK	Counter 1 Clock
5	CTR1 OUT	Counter 1 Output, programmable gain control for AIM-16 (mid bit)
6	CTR2 OUT	Counter 2 Output, programmable gain control for AIM-16 (MSB)
7	OP0	LSB Digital Output, sub-multiplexer channel select
8	OP1	Bit 1 Digital Output, sub-multiplexer channel select
9	OP2	Bit 2 Digital Output, sub-multiplexer channel select
10	OP3	MSB Digital Output, sub-multiplexer channel select
11	DIG COM	Power (Digital) ground
12	LL GND	Low Level (Analog) Ground
13	LL GND	Low Level (Analog) Ground
14	LL GND	Low Level (Analog) Ground
15	LL GND	Low Level (Analog) Ground
16	LL GND	Low Level (Analog) Ground
17	LL GND	Low Level (Analog) Ground
18	LL GND	Low Level (Analog) Ground
19	VREF	+10.0VDC (220mA) A/D reference output
20	-12VDC	-12VDC Power from the Computer Bus
21	CTR0 GATE	Counter 0 Gate
22	CTR1 GATE	Counter 1 Gate
23	CTR2 GATE	Counter 2 Gate
24	INT	Interrupt input, positive edge trigger
25	IP1	Digital Input, Bit 1
26	IP2	Digital Input, Bit 2
27	IP3	Digital Input, Bit 3
28	DIG COM	Power (Digital) ground
29	+5VDC	+5VDC Power from the Computer Bus
30	CH7 IN	Chl 7 Analog Input
31	CH6 IN	Chl 6 Analog Input
32	CH5 IN	Chl 5 Analog Input
33	CH4 IN	Chl 4 Analog Input
34	CH3 IN	Chl 3 Analog Input
35	CH2 IN	Chl 2 Analog Input
36	CH1 IN	Chl 1 Analog Input
37	CH0 IN	Chl 0 Analog Input

Table B-1: AIO8 Output Connector Pin Assignments

Appendix C: Basic Integer Variable Storage

Data are stored in integer variables (% type) in 2's complement form. Each integer variable uses 16 bits or two bytes of memory. Sixteen bits of binary data is equivalent to 0 to 65,535 decimal but the 2's complement convention interprets the most significant bit as the sign bit so the actual range is -32,768 to +32,767. Numbers are represented as follows:

Number	High Byte								Low Byte							
	B7	B6	B5	B4	B3	B2	B1	B0	B7	B6	B5	B4	B3	B2	B1	B0
+32767	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
+10000	0	0	1	0	0	1	1	1	0	0	0	1	0	0	0	0
+1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
-1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1
-10000	1	1	0	1	1	0	0	0	1	1	1	1	0	0	0	0
-32768	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Note: Bit 7 (B7) of the high byte is the sign bit. (1=negative, 0=positive)

Integer variables are the most compact form of storage for the 12-bit data from the A/D converter and the 16-bit data from the interval timer. Therefore, to conserve memory and disk space and to optimize execution time, all data exchange via the CALL is through integer type variables.

This poses a programming problem when handling unsigned numbers in the range 32,768 to 65,535. If you wish to input or output an unsigned integer greater than 32,767, then it is necessary to work out what its 2's complement signed equivalent is. For example, if 50,000 decimal is to be loaded into a 16-bit counter, an easy way to convert this to binary is to enter BASIC and execute PRINT HEX$(50000). This returns C350 which, in binary form is: 1100 0011 0101 0000. Since the most significant bit is a one, this would be stored as a negative integer and, in fact, the correct integer variable value would be 50,000 - 65,536 = -15,536.

Thus, the programming steps to switch between integer and real variables for representation of unsigned numbers between 0 and 65,535 is:

-From real variable N (where 0 <= N <= 65,535) to integer variable N%:

```
xxx10 IF N<=32767 THEN N% = N ELSE N% = N-65536
```

-From integer variable N% to real variable N:

```
xxx20 IF N% >= 0 THEN N=N% ELSE N = N%+65536
```

Customer Comments

If you experience any problems with this manual or just want to give us some feedback, please email us at: ***manuals@accesioproducts.com.***. Please detail any errors you find and include your mailing address so that we can send you any manual updates.

10623 Roselle Street, San Diego CA 92121
Tel. (619)550-9559   FAX (619)550-7322
www.accesioproducts.com