INDUSTRIAL COMPUTER SOURCE®

Model AIO8-P
Product Manual

MANUAL NUMBER : 00650-013-13B

INDUSTRIAL COMPUTER SOURCE® http:/www.indcompsrc.com

6260 SEQUENCE DRIVE, SAN DIEGO, CA 92121-4371 (619) 677-0877 (FAX) 619-677-0895

INDUSTRIAL COMPUTER SOURCE EUROPE *TEL (1) 69.18.74.40 FAX (1) 64.46.40.42* • ***INDUSTRIAL COMPUTER SOURCE (UK) LTD*** *TEL 01243-533900 FAX 01243-532949*

FOREWARD

This product manual provides information to install, operate and or program the referenced product(s) manufactured or distributed by Industrial Computer Source. The following pages contain information regarding the warranty and repair policies.

Technical assistance is available at: **1-800-480-0044**.

Manual Errors, Omissions and Bugs: A "Bug Sheet" is included as the last page of this manual. Please use the "Bug Sheet" if you experience any problems with the manual that requires correction.

NOTE

The information in this document is provided for *reference* only. Industrial Computer Source 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 Industrial Computer Source, nor the rights of others.

This page intentionally left blank

Guarantee

A thirty day money-back guarantee is provided on all **standard** products sold. **Special order products** are covered by our Limited Warranty, *however they may not be returned for refund or credit*. *EPROMs, RAM, Flash EPROMs or other forms of solid electronic media are not returnable for credit - but for replacement only. Extended Warranty available. Consult factory.*

Refunds

In order to receive refund on a product purchase price, the product must not have been damaged by the customer or by the common carrier chosen by the customer to return the goods, and the product must be returned complete (meaning all manuals, software, cables, etc.) within 30 days of receipt and in as-new and resalable condition. The **Return Procedure** must be followed to assure prompt refund.

Restocking Charges

Product returned *after* 30 days, and *before* 90 days, of the purchase will be subject to a **minimum** 20% restocking charge and any charges for damaged or missing parts.

Products not returned within 90 days of purchase, or products which are not in as-new and re-saleable condition, are not eligible for credit return and will be returned to the customer.

Limited Warranty

One year limited warranty on all products sold with the exception of the "Performance Series" I/O products, which are warranted to the original purchaser, for as long as they own the product, subject to all other conditions below, including those regarding neglect, misuse and acts of God. Within one year of purchase, Industrial Computer Source will repair or replace, at our option, any defective product. At any time after one year, we will repair or replace, at our option, any defective "Performance Series" I/O product sold. This does not include products damaged in shipment, or damaged through customer neglect or misuse. Industrial Computer Source will service the warranty for all standard catalog products for the first year from the date of shipment. After the first year, for products not manufactured by Industrial Computer Source, the remainder of the manufacturer's warranty, if any, will be serviced by the manufacturer directly.

The **Return Procedure** must be followed to assure repair or replacement. Industrial Computer Source will normally return your replacement or repaired item via UPS Blue. *Overnight delivery or delivery via other carriers is available at additional charge.*

The limited warranty is void if the product has been subjected to alteration, neglect, misuse, or abuse; if any repairs have been attempted by anyone other than Industrial Computer Source or its authorized agent; or if the failure is caused by accident, acts of God, or other causes beyond the control of Industrial Computer Source or the manufacturer. Neglect, misuse, and abuse shall include any installation, operation, or maintenance of the product other than in accordance with the owners' manual.

No agent, dealer, distributor, service company, or other party is authorized to change, modify, or extend the terms of this Limited Warranty in any manner whatsoever. Industrial Computer Source reserves the right to make changes or improvements in any product without incurring any obligation to similarly alter products previously purchased.

Shipments not in compliance with this Guarantee and Limited Warranty Return Policy will not be accepted by Industrial Computer Source.

Return Procedure

For any Limited Warranty or Guarantee return, please contact Industrial Computer Source's Customer Service at **1-800-480-0044** and obtain a Return Material Authorization (RMA) Number. All product(s) returned to Industrial Computer Source for service or credit **must** be accompanied by a Return Material Authorization (RMA) Number. Freight on all returned items **must** be prepaid by the customer who is responsible for any loss or damage caused by common carrier in transit. Returns for Warranty **must** include a Failure Report for each unit, by serial number(s), as well as a copy of the original invoice showing date of purchase.

> To reduce risk of damage, returns of product must be in an Industrial Computer Source shipping container. If the original container has been lost or damaged, new shipping containers may be obtained from Industrial Computer Source Customer Service at a nominal cost.

Limitation of Liability

In no event shall Industrial Computer Source be liable for any defect in hardware or software or loss or inadequacy of data of any kind, or for any direct, indirect, incidental, or consequential damages in connection with or arising out of the performance or use of any product furnished hereunder. Industrial Computer Source liability shall in no event exceed the purchase price of the product purchased hereunder. The foregoing limitation of liability shall be equally applicable to any service provided by Industrial Computer Source or its authorized agent.

Some *Sales Items* and *Customized Systems* are **not** subject to the guarantee and limited warranty. However in these instances , any deviations will be disclosed prior to sales and noted in the original invoice. ***Industrial Computer Source reserves the right to refuse returns or credits on software or special order items.***

Table of Contents

List of Figures

List of Tables

Current Revision 13B

August 1997

This page intentionally left blank

Chapter 1: Functional Description

The AIO8-P 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-P card.

Analog Inputs

The card accepts eight single-ended analog input channels. In the case of the AIO8-P, the full scale input for all channels is ±5V (0.00244V resolution). Inputs are single-ended with a common ground and can withstand overvoltages 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 eight analog inputs are DIP-switch selectable as either differential or single-ended inputs. The analog-to-digital converter (A/D) is a 12-bit successive-approximation type with a sample and hold input. Conversion time is typically 25µSec, (35µSec, maximum) and, depending on the speed of the software and computer platform, throughputs of up to 30,000 conversions per second are attainable.

Input System Expansion

The AIO8-P card may be used alone or it can support up to eight AT16-P or LVDT8-P analog input expansion cards. An 4-bit standard LSTTL logic output from the AIO8-P is used to select one of 16 analog input channels at the AT16-P. When interfacing to the LVDT8-P, three bits are used to select one of eight LVDT8-P inputs. Since the eight-input multiplexer on the AIO8-P 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-P 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-P. 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-P includes a type 8253-5 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 may be initiated by the Counter/Timer by installing a jumper on the I/O connector Also, software provided by Industrial Computer Source includes the means to use these counters to provide programmable gain commands to the instrumentation amplifier on the AT16-P expansion sub-multiplexer card. Refer to Chapter 7: Programmable Interval Timer for a description of the functions of the 8253 counter/timer chip.

Interrupts

Interrupts are supported from external inputs. 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 AT16-P expansion card, or the AIO8-P multiplexer.

Utility Software

Utility software included with the AIO8-P is provided on a diskette. 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(SETAIO8), the second setup program (SETMUX) is used when an AT16-P and / or LVDT8-P is used in conjunction with the AIO8-P. In the case of the AT16-P, gains are assignable on a channel-by-channel basis. Linearization for all the commonly used thermocouple types as well as for platinum RTD's is also menu selectable.

A driver configuration file is generated or modified by the setup program and can be used to configure the AIO8-P driver. Chapter 2: Software Installation describes the format of this configuration file. This driver has 17 Tasks as will be described in detail in Chapter 5: AIO8-P Driver Reference of this manual.

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

Enhancements

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

A. UTB-K Screw Terminal Cards.

The UTB-K universal termination assembly provides screw terminals to facilitate field wiring to the AIO8(G)-P I/O boards. The UTB-K includes a metal enclosure to protect field terminations from environmental factors and to provide an easy mounting method for the termination assembly.

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. AT16-P Expansion Multiplexer and Instrumentation Amplifier

The AT16-P 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 AT16-P allows multiplexing of 16 analog input signals to a single AIO8-P analog input channel. As described earlier, up to eight AT16-P's can be connected to a single AIO8-P to provide input capability for up to 128 analog inputs.

The AT16-P 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 AT16-P or, alternatively, may be jumpered to an unused AD12-8 input channel. Open-thermocouple or "break detect" circuitry is provided.

The AT16-P may also be used with 3-wire RTD's (AT16-PR), strain gages, and 4-20mA current transmitter inputs. In this latter case, an application-specific version, the AT16-PI, is available.

C. LVDT8-P Multiplexer and Interface Card

This card provides AC excitation and signal conditioning to eight independent LVDT transducers. As many as eight LVDT8-P's may be connected to an AIO8-P to accommodate up to 64 transducers.

D. LABTECH NOTEBOOK

Labtech Notebook is a menu-driven data acquisition software package. It is capable of foreground or background operation. Additional capabilities including ICON setup, variable sample rates on a channel-by-channel basis, additional mathematical and statistical calculations, and compatibility with expanded memory.

Specifications

ANALOG INPUTS

Channels

8 single-ended inputs with common ground.

Voltage Range

±5V

Resolution

12 binary bits.

Accuracy

±0.05% of reading ±1 LSB.

Input Impedance

10Mega-ohms or 125nA at 25° C.

Overvoltage

±30VDC

Linearity

±1 LSB.

Temperature. Coefficient

±10 µV/°C zero stability
±25 µV/°C gain stability

Common Mode Rejection (when used with AT16-P)

90 dB when gain = 1
125 dB when gain = 100

Throughput

30,000 conversions per second maximum.

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 0.8 VDC at -0.4mA 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

8253-5 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 10MHz.

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 to 60°C.

Storage Temp

-40° to 100°C.

Humidity

0 to 90% RH, non-condensing.

Size

7.0 inches long, requires full-size slot.

Power Required

+5VDC: 320mA maximum
+12VDC: 10mA maximum
-12VDC: 15mA maximum

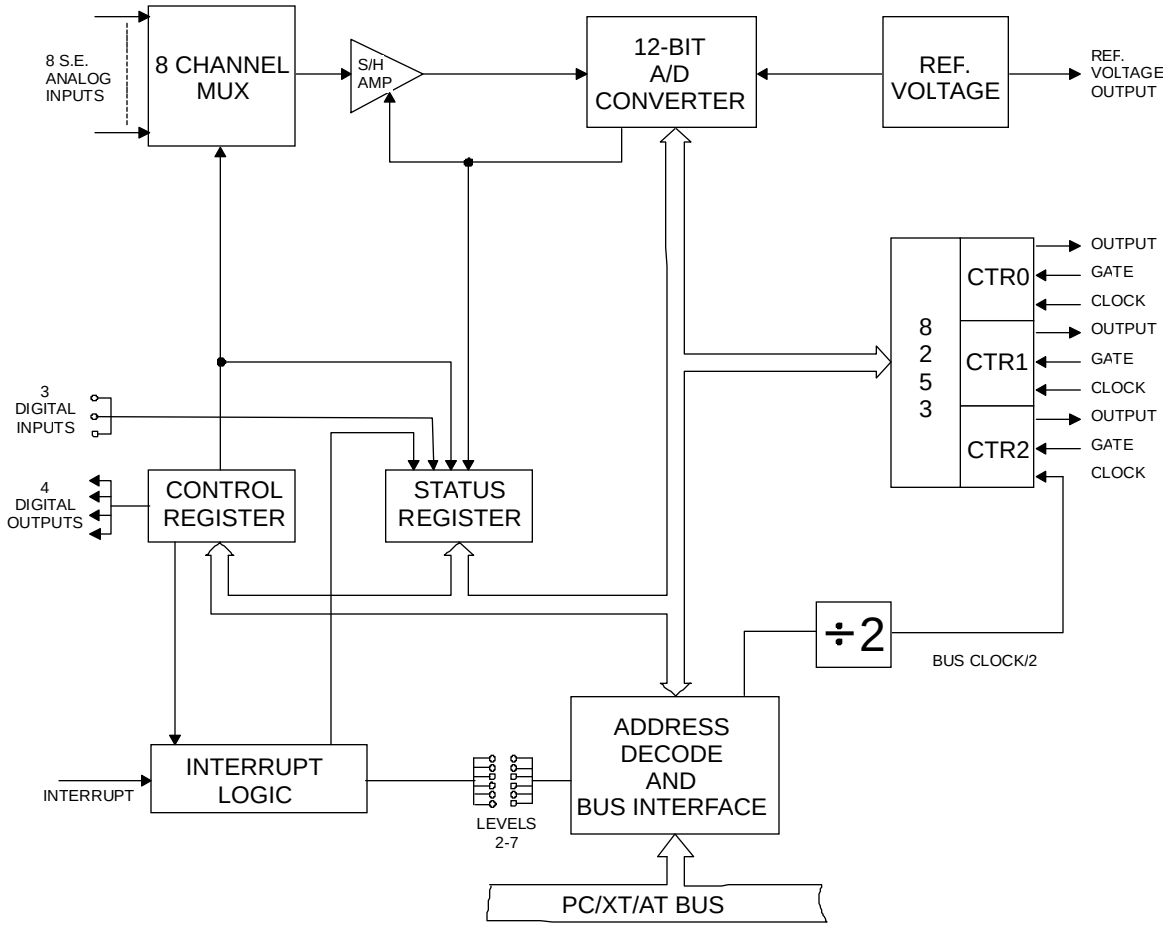


Figure 1-1: AIO8-P Block Diagram

How to remain CE Compliant

In order for machines to remain CE compliant, only CE compliant parts may be used. To keep a chassis compliant it must contain only compliant cards, and for cards to remain compliant they must be used in compliant chassis. Any modifications made to the equipment may affect the CE compliance standards and should not be done unless approved in writing by Industrial Computer Source.

The Model AIO8-P is designed to be CE Compliant when used in an CE compliant chassis. Maintaining CE Compliance also requires proper cabling and termination techniques. The user is advised to follow proper cabling techniques from sensor to interface to ensure a complete CE Compliant system. Industrial Computer Source does not offer engineering services for designing cabling or termination systems. Although Industrial Computer Source offers accessory cables and termination panels, it is the user's responsibility to ensure they are installed with proper shielding to maintain CE Compliance.

This page intentionally left blank

Chapter 2: Software Installation

Software Provided

The following utility software is provided with AIO8-P in MS-DOS format on a floppy disk:

A. A menu-driven setup and calibration program **SETAIO8.EXE**.

B. A menu-driven setup and calibration program for the sub-multiplexer cards, **SETMUX.EXE**.

C. A standard driver program provided in three forms; a BASIC loadable file called **AIO8DRV.BIN**, a "C" language linkable file called **AIO8DRVC.OBJ**, and a QuickBASIC and Pascal linkable file called **AIO8DRV.OBJ**. All forms of the driver were created using Turbo Assembler, by Borland.

D. A program to locate available address space, **FINDBASE.EXE**

E. Sample programs in BASIC, QuickBASIC, "C", Pascal, and a VisualBasic DLL.

F. A utility program that allows you to generate up to a 10th order polynomial approximation, **POLY.EXE** (refer to Appendix A)

Hard Disk Installation

Installation Program

The software should be installed before the card is physically installed in the chassis. A setup routine titled **SETAIO8.EXE**, describes how to set all the address switches and jumpers on the card. Each of the settings is also described in its appropriate section of this manual.

The AIO8-P Software Package utilizes compression schemes to simplify installation and to permit the use of a single diskette for shipment. A program is provided on your master disk to copy and expand the AIO8-P Software Package onto your hard-drive. To begin the installation, place the AIO8-P master diskette in a floppy drive and execute the INSTALL.EXE program. For example, if you have placed the master disk in floppy drive A, you would type A:INSTALL to execute the installation program.

The installation program will ask you for various installation options, and will provide default settings. These default settings may be selected (by pressing ENTER) if they will work with your particular application and system setup, or respond to the questions with appropriate answers as needed.

When all of the installation options have been set, the program will expand the AIO8-P program files into the destination you have selected. Once this process is complete, please put your Industrial Computer Source AIO8-P Master Diskette in a safe place as backup.

The installation process will create the following directory structure on your destination disk:

AIO8-P Contains the SETAIO8.EXE setup and calibration program, the SETMUX.EXE sub-multiplexer card setup and calibration program, and POLY.EXE, the linearization polynomial program.

PSAMPLES Contains Pascal samples and the Pascal-linkable driver. Sub-directory of AIO8-P.

CSAMPLES Contains "C" samples and the "C"-linkable driver. Sub-directory of AIO8-P.

BSAMPLES Contains the BASIC and QuickBASIC samples as well as the binary and linkable drivers. Sub-directory of AIO8-P.

Findbase Routine

One of the programs included in the installation is a routine titled **FINDBASE.EXE.** This program can be used to find an unused section of I/O memory to assign to the AIO8-P. It simplifies base address selection. The program will scan your computer's I/O ports for available locations which would be suitable for the card. The program asks you to pick the number of address bytes required from the supplied list. In this case, the AIO8-P requires 8 address bytes so select 8 from the list. It will then present the first address location with that much space available. The instructions are self explanatory. A text file, **FINDBASE.TXT** contains more information on its use.

Configuration File

The Configuration File has several purposes; most of which are associated with use of the software drivers provided with your AIO8-P 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 Industrial Computer Source's 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:

- AIO8-P plug in PC card.
- AT16-P sub-multiplexer card.
- LVDT-8 sub-multiplexer card.
- Programmable gain assignments.
- 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 AT16-P, the counters will not be available for general use. The counters are required by the drivers to set the gain on the AT16-P.

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 AT16-P 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 AT16-P 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 RTD's 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 AT16-P 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 Chapter 5: AIO8-P Driver Reference. The AT16-P 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-P 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-P 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-P has an AT16-P sub-multiplexer card attached. The unit of measure for this card is °F. The AT16-P 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-P has an LVDT8-P attached, with all eight channels unskipped.

D. Channel 2 of the AIO8-P is a direct channel that is unskipped.

E. All other A/D channels are direct and skipped.

This page intentionally left blank

Chapter 3: Hardware Installation

Before installing the card, be sure to install the software as described in Chapter 2, and run the **SETAIO8.EXE** program. Check the appropriate sections of this manual for further information on address and option selection.

To install the card:

1. Perform the software installation as described in Chapter 2.
2. Run **FINDBASE.EXE** if required.
3. Turn off computer power.
4. Remove the computer cover.
5. Remove the blank I/O backplate.
6. Set the interrupt option jumpers as desired.
7. Set the Base Address.
8. Install the card in an I/O expansion slot. Attach cable to card.
9. Inspect for proper fit of the card and cable. Tighten the screws.
10. Replace the computer cover and apply power.

Option Selection

The AIO8-P 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 on diskette 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.

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.

Base Address

The following section shows you how to select and set the address.

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. If you are uncertain of your avaialbe space, run the **FINDBASE** utility provided on the included diskette. Refer to the Findbase section of Chapter 2 for further information.

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.

Hex Range	Usage
000-1FF	Internal System - Not Usable
200-20F	Game Control
210-217	Expansion Unit
220-24F	Reserved
278-27F	Reserved
2E8-2EF	Serial Port
2F0-2F7	Reserved
300-31F	Asynchronous Communications (secondary)
320-32F	Prototype Card
378-37F	Fixed Disk
380-38C	Printer
3A0-3A9	SDLC Communications
3B0-3BF	Binary Synchronous Communications (primary)
3C0-3CF	Reserved
3DO-3DF	Color/Graphics
3E0-3E7	Reserved
3E8-3EF	Serial Port
3F0-3F7	Diskette
3F8-3FF	Asynchronous Comunications (primary)

Table 3-1: Standard Base Address Assignments

Setting the Base Address

The AIO8-P 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 AIO8-P. The following procedure will show you how to set the base address. See Table 3-2: Base Address Example, 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 FIGURE 3-3 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 on diskette with AIO8-P 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 **SETAIO8** 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.

Calibration and Test

All Industrial Computer Source cards are calibrated prior to shipment. However, periodic calibration of AIO8-P 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 SETAIO8 program on the diskette 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-P 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 SETAIO8 program.

1) Start the calibration program by typing **SETAIO8** 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

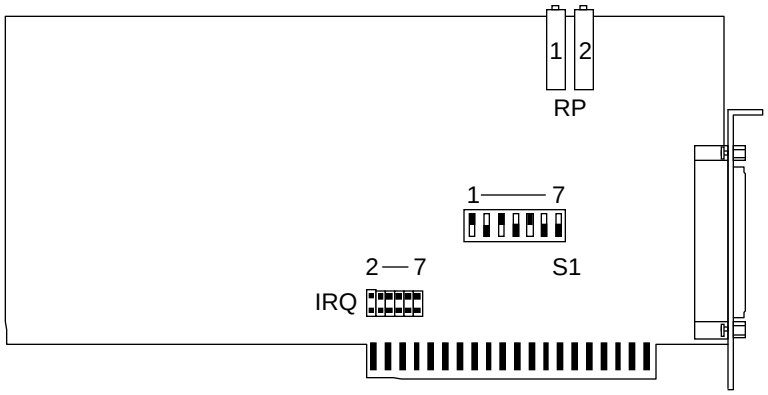
6) Use the arrow key to select option **2. Set Gain**, from the **Action Menu**.

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.



Switch
S1 = Base Address

Potentiometers
RP1 = Gain Adjust
RP2 = Bipolar Offset

Jumpers
IRQ1 - IRQ7 = Select IRQ Level

Figure 3-1: AIO8-P Option Selection Map

Chapter 4: Programming the AIO8-P

This chapter provides you with information on how to program the AIO8-P. 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-P. If you plan to use the Industrial Computer Source provided driver, refer to the section in this chapter, **Programming Using the Driver**, and to **Chapter 5: AIO8-P 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-P 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-P. You may find it easer to design your program using the supplied drivers.

AIO8-P Register Address Map

The AIO8-P 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	Control Counter

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 AT16-P, or LVDT8-P sub-multiplexer cards.

-IEN: This bit enables/disables AIO8-P 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 mutiplexer channel address on the AIO8-P 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
BOC	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-P card sets this bit high(1). It is reset to state 0 by a computer reset, or a write to the control register.

-MA3-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 a 12-bit A/D conversion 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 eight bits of a 12-bit A/D conversion output in binary form, or the results of an 8-bit conversion.

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

- AD4-AD11: The upper eight 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 (-.5 Scale)
:	:	:
1000 0000 0000	800	±0V
1000 0000 0001	801	+0.0024V
:	:	:
1100 0000 0000	C00	+2.5000V (+.5 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 8253 counters is presented in **Chapter 7: Programmable Intrerval Timer** section of this manual. However, for a full description of features of this extremely versatile IC, refer to the Intel 8253 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	Illegal

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

RW1	RW0	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-P is straightforward but the coding can be rather tedious. To assist you in building your application quickly, Industrial Computer Source provides a driver. 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 **Chapter 5: AIO8-P 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 AT16-P in programmable 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 **Chapter 5: AIO8-P Driver Reference**.

PARAMETERS This is an array of integers which contains information required by the driver.

Chapter 5: AIO8-P 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.

```
int      task,params[5],status;
```

C. Make your assignment to these variables as desired for the function you wish to perform. See **Chapter 5: AIO8-P 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 Chapter 5: AIO8-P 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 Chapter 5: AIO8-P 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 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 Chapter 5: AIO8-P Driver Reference for details on each task.

E. Make the call to the driver.

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

Using the Driver with Visual Basic

Included with the supplied software is a DLL (Dynamic Link Library) called **VBACCES.DLL.** It is compatible with Visual Basic version 3.0. **VBACCES.DLL** must be copied to your Windows directory. Also included is a sample program to help you interface this DLL with Visual Basic. The program is titled **VBACCES.FRM**, and its global definition file is **VBACCES.GBL**. The information in the **.GBL** file must be contained in any application that uses the DLL, but does not have to be in a separate file. A project file **VBACCES.MAK** is also included.

The commands provided are:

OutPort, Outportb: Allows write access to the I/O bus, similar to the C language outport and outportb functions.

InPort, InPortb: Allows read access to the I/O bus, similar to the C language inport and inportb functions.

Peek, Poke: Allows read and write access to RAM, similar to BASIC's Peek and Poke statements.

Please refere to the **VBACCES.GBL** file for programming information related to the above function.

Chapter 5: AIO8-P Driver Reference

This chapter provides detailed information on the functions available from the AIO8-P driver. The chapter 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-P and an AT16-P or LVDT8-P(if used). The first 16 point addresses (0-15) refer to the AT16-P channels for an AT16-P attached to channel 0 of the AIO8-P. The second 16 point addresses (16-31) refer to the 16 channels of an AT16-P attached to channel 1 of the AIO8-P, and so on. Thus, with eight single ended A/D channels, a point address may be as large as 127.

If your are using LVDT8-Ps, then the first eight point address (0-7) refer to the LVDT8-P attached to channel 0 of the AIO8-P. 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 LVDT8-P 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 AT16-P and/or LVDT8-P 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 AT16-P. 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 AT16-P. When using this feature, the counters are not available, as they are used to set the AT16-P 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-P combined with the AT16-P 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-P, with none for the AT16-P or LVDT8-P (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 SETAIO8-P 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 AT16-P or LVDT8-P 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]:  AT16-P mode
   0 = AT16-P using programmable gains
   1 = AT16-P 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 AT16-P mode.

status = 20: Error opening configuration file.

status = 21: Error reading configuration file.

status = 22: Invalid configuration file data.

Example:

```
int      task,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;                   /* AT16-P 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:

```
int      task,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 AT16-P 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:	AT16-P in manual mode.

Example:

```
int      task,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:

```
int      task,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	AT16-P 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 AT16-P is being used, and the driver has been confgured 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:	AT16-P in manual mode.

Example:

```
int      task,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
                                       /* 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 AT16-Ps or LVDT8-Ps 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:

```
int      task,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 AT16-Ps */
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 AT16-P OR LVDT8-P 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:

```
int      task,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:

```
int      task,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:

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

int      datbuf[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 AT16-P'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.
               params[5]:  A/D trigger mode.
               0:    Start A/D on each positive transition
               of the IP0/TRG0 pin.
               1:    Use counters 1 and 2 to supply the A/D
               trigger.
               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:

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

int      datbuf[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 AT16-P on board sensor is desired, then assign this sensor to the point list as the first channel of a given AT16-P (ie. 0, 16,32,48 etc). Make sure that the TMP jumpers are installed on the AT16-P. Finally, assign the curve "T" to this point address using SUBTASK 2. Any other point addresses on the AT16-P will now be junction compensated automatically by the driver each time a point address is converted.

4) The reference junction circuit on the AT16-P card generates 24.4 mV/°C. The counts read in at a gain of 1 are 2.44 millivolts/count. Thus, each count represents 0.1°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 AT16-P. 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 °F.
   params[4]:  temperature in tenths of °C.
```

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:

```
int      task,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
                                          AT16-P */
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 AT16-P. 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:

```
int      task,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 AT16-P is being used, this task should not be called, as these digital output bits are used to set the channel on the AT16-P.

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:

```
int      task,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:

```
int      task,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 Chapter 7: Prgramming Interval Timer.

2) If the AT16-P 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 AT16-P 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: AT16-P programmable mode set.

Example:

```
int      task,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 Chapter 7: Programmable Interval Timer.

2) Counter/timer is latched before read.

3) If the AT16-P 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 AT16-P 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:

```
int      task,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 AT16-P 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 AT16-P 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) The 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 multiples
```

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:	AT16-P in programmable gain mode.

Example:

```
int      task,params[7],status;        /* these are globally declared
                                         variables */double frequency;
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 AT16-P 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 AT16-P 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:	AT16-P is in programmable gain mode.

Example:

```
int      task,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;
```

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 AT16-P mode: AT16-P 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 AT16-P to be in programmable gain mode has been called when the AT16-P was set for manual gain mode.

19: An attempt to use a task that requires the AT16-P to be in manual gain mode has been called when the AT16-P was set for programmable gain mode.

20: Error opening configuration file.

21: Error reading configuration file.

22: Configuration file data are not correct.

This page intentionally left blank

Chapter 6: A/D Converter Applications

Connecting Analog Inputs

The AIO8-P 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-P, the AT16-P multiplexer card must be added. The AT16-P supplies 16 channels of differential input.

Thus, if the signal source has one side connected to a local ground, the AIO8-P/AT16-P 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-P/AT16-P 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-P/AT16-P 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.

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 250Ù shunt resistor across the A/D input terminals. The resultant 1-5V signal can then be measured. The Industrial Computer Source UTB-K screw terminal accessory board, for example, includes a breadboard area with plated through holes that allow insertion of shunt resistors.

If an AT16-P multiplexer expansion card is being used, pre-wired pads are provided on the AT16-P. If all the inputs are 4-20mA range current inputs from current transmitters, then there is a configuration of the multiplexer expansion board called AT16-PI. 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-P. Industrial Computer Source’s AT16-P provides capability for 16 channels per input plus a common instrumentation amplifier. Up to eight AT16-P’s can be added to one AIO8-P 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. Industrial Computer Source cannot accept liability for this kind of accident. 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-P. 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-P card.

This page intentional;ly left blank

Chapter 7: Programmable Interval Timer

The AIO8-P contains a type 8253 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 8253 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-P 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 1/2 multiple of the PC bus clock. The output and gate of counter #2 is also available at the I/O connector. If the AT16-P is being used with programmable gain, then all counters are required for setting the gains on the AT16-P.

Operational Modes

The 8253 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 8253 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 8253 :

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 countdown 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-P, the 8253 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	Illegal

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

RW1	RW0	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 + 7,0x00);  /* 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 AIO8-PDRV Driver

In practice, TASKS 14 and 15 of the AIO8-PDRV 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;                          /* readcounter 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 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:

$$f(x) = C(0) + C(1)x^1 + C(2)x^2 + C(3)x^3 + C(4)x^4 + C(5)x^5$$

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 Gaussian Elimination with partial pivoting to improve accuracy.

Appendix B: Cabling and Connector Information

Connections are made to the AIO8-P 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 such as Industrial Computer Source part number UTB-K. The pin assignments are as follows:

Pin #	Name	Function
1	+12VDC	+12VDC from the computer Bus
2	CTR0 Clk	Counter 0 Clock
3	CTR0 Out	Counter 0 Output, programmable gain control for AT16-P (LSB)
4	CTR1 Clk	Counter 1 Clock
5	CTR1 Out	Counter 1 Output, programmable gain control for AT16-P (mid bit)
6	CTR2 Out	Counter 2 Output, programmable gain control for the AT16-P (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

(continued on next page)

Pin #	Name	Function
19	VRef	+10.0VDC (220mA) A/D reference output
20	-12VDC	-12VDC 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 from the computer Bus
30	CH7 In	Channel 7 Analog Input
31	CH6 In	Channel 6 Analog Input
32	CH5 In	Channel 5 Analog Input
33	CH4 In	Channel 4 Analog Input
34	CH3 In	Channel 3 Analog Input
35	CH2 In	Channel 2 Analog Input
36	CH1 In	Channel 1 Analog Input
37	CH0 In	Channel 0 Analog Input

Appendix C: Base 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							
	B7	B6	B5	B4	B3	B2	B1	B0
+32767	0	1	1	1	1	1	1	1
+10000	0	0	1	0	0	1	1	1
+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
-10000	1	1	0	1	1	0	0	0
-32767	1	0	0	0	0	0	0	0

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

Declaration of Conformity

INDUSTRIAL COMPUTER SOURCE®

6260 Sequence Drive
San Diego, CA 92121-4371
(800) 523-2320

Industrial Computer Source declares under its own and full responsibility that the following products are compliant with the protection requirements of the 89/336/EEC and 73/23/EEC directives.

Only specific models listed on this declaration and labeled with the CE logo are CE compliant.

AIO8-P

Conformity is accomplished by meeting the requirements of the following European harmonized standards:

EN 50081-1:1992 Emissions, Generic Requirements.

-EN 55022 Measurement of radio interference characteristics of information technology equipment.

EN 50082-1:1992 Immunity, Generic Requirements.

-IEC 801-2:1984 Immunity for AC lines, transients, common, and differential mode.
-IEC 801-3:1984 Immunity for radiated electromagnetic fields.
-IEC 801-4:1988 Immunity for AC and I/O lines, fast transient common mode.

EN 60950:1992 Safety of Information Technology Equipment.

Information supporting this declaration is contained in the applicable Technical Construction file available from:

INDUSTRIAL COMPUTER SOURCE EUROPE®

Z.A. de Courtaboeuf
16, Avenue du Québec
B.P. 712
91961 LES ULIS Cedex

August 28, 1997
San Diego, CA

Mr. Steven R. Peltier
President & Chief Executive Officer

BUG REPORT

While we have tried to assure this manual is error free, it is a fact of life that works of man have errors. We request you to detail any errors you find on this BUG REPORT and return it to us. We will correct the errors/problems and send you a new manual as soon as available. Please return to:

INDUSTRIAL COMPUTER SOURCE®

Attn: Documentation Department
P. O. Box 910557
San Diego, CA 92121-0557

Your Name: _____

Company Name: _____

Address 1: _____

Address 2: _____

Mail Stop: _____

City: _____ State: _____ Zip: _____

Phone: (_____) _____

Product: **AIO8-P**

Manual Revision: **00650-013-13B**

Please list the page numbers and errors found. Thank you!