Model AIO8G-P
Product Manual

MANUAL NUMBER : 00650-002-35B

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 35B

August 1997

Chapter 1: Introduction

The AIO8G-P is a multifunction, moderate-speed, analog/digital I/O card with counter/timers. These cards may be used with IBM PC/XT/AT and 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.

Analog Inputs

The card accepts eight analog inputs. The inputs are DIP-switch selectable as either differential or single-ended. The card can withstand a continuous overload of ±30 volts and brief transients of several hundred volts. All inputs are fail-safe; i.e. open circuit when the power is off.

The AIO8G-P includes a programmable-gain amplifier which provides a total of nine software selectable analog voltage input ranges. The analog-to-digital converter (A/D) is a 12-bit, successive-approximation type with sample and hold input. Conversion time is typically 25 microseconds (35 microseconds max.) and, depending on the software and the computer, throughput rates of up to 30,000 channels/sec are attainable.

Analog Input Expansion

The AIO8G-P cards support up to eight AT16-P, or eight LVDT8-P analog input expansion cards. A 4-bit standard LSTTL logic output from the AIO8G-P is used to select one of 16 analog input channels at the AT16-P and a 3-bit logic output is used to select one of eight analog input channels at the LVDT8-P. To accommodate more than 128 inputs, a second AIO8G-P (and companion input expansion cards) can be used.

Counter/Timer

An 8254 Counter/Timer chip is included along with a dedicated 1 MHz crystal oscillator. This chip contains three separate 16-bit down counters and can be used to generate periodic interrupt requests, for event counting, pulse and waveform generation, frequency and period measurement, etc. Refer to Chapter 5 of this manual for a description of this chip's capabilities. Timed 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 means to use these counters to provide programmable gain commands to the instrumentation amplifier on the AT16-P expansion multiplexer.

Interrupts

Interrupts are supported by external input signals from the A/D converter. Selection of interrupt levels IRQ2 through IRQ7 is made by jumper. Interrupts are software enabled and disabled. An interrupt request may be canceled by either of the following signals:

1. Computer reset signal.
2. Writing a command word to the card. That is either updating the channel selection on the AT16-P expansion card, or on the AIO8G-P multiplexer.

Utility Software

Utility software provided with the card includes a menu-driven setup and calibration program, and a device driver in assembly language as well as in binary object code. Two setup programs are provided; one for the card itself and one for the AT16-P expansion multiplexer. In this latter case, gains are assignable on a channel-by-channel basis. Finally, sample programs in QuickBASIC, C, and Pascal are provided.

Enhancements

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

1. UTB-K Screw Terminal Boards

   The UTK universal termination assembly provides screw terminals to facilitate fields 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.

2. 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 AIO8G-P analog input channel. As described in Paragraph 1.1.2, up to eight AT16-P's can be connected to a single AIO8G-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, 400, 500, and 1000. In addition, these gains can be software programmed to provide individual channel gains.

   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 AIO8G-P input channel. Open-thermocouple or "break detect" circuitry is provided.

   The AT16-P may also be used with 3-wire RTD's, strain gages, and 4-20 mA current transmitter inputs. In this latter case, an application-specific version, the AT16DCI-P, is available.

3. LVDT8-P Multiplexer and Interface Card for LVDT's: This card provides AC excitation and signal conditioning to eight independent LVDT transducers. As many as eight LVDT8-P's can be connected to an AIO8G-P to accommodate as many as 64 transducers.

Specifications

Analog Inputs

No. of Channels	Eight differential or single-ended (DIP switch selectable).
Voltage Range:	Programmable ±5V (default) ±10V ±0.5V ±0.01 0-10V 0-1V, 0-0.1V, 0-0.02V
Overvoltage Protection:	±30VDC.
Input Impedance:	10 Megohm or 125 nA at 25° C.
Overall Accuracy:	±0.05% of reading ±1 LSB.
Linearity:	±1 LSB.
Resolution:	12 bit binary.
Temperature Coefficient:	±10µV/° C zero stability. ±25µV/° C gain stability.
Common Mode Rejection:	90 db when gain = 1 125 db when gain = 100
Throughput:	30,000 conversions per second maximum.
A/D Conversion Time:	35 µS Max, 25 µS typical

Digital Inputs/Outputs

Inputs	Logic Low: 0 to 0.4 V at 8 mA source current Logic High: 2.4 to 5.0 V at 0.4 mA source current
Outputs	Logic Low: 0 to 0.4 V at 8 mA sink current. Logic High: 2.4 to 5.0 V at 0.4 mA source current.

Counter/Timer

Type:	82C54 programmable interval timer, three 16-bit down counters
Drive capability:	5 LSTTL loads (2.2 mA at 0.45VDC)
Input Load (Gate and Clock):	±10 µA, TTL/CMOS compatible
Input Clock Frequency:	10 MHz max.
Active Count Edge:	Negative edge
Clock Pulse Width:	50µS high/50 µS low min.

Interrupts

Level:	Jumper selectable, levels 2-7.
Enable/Disable:	Via software, (INTE bit of Control Register)
Source:	External, positive edge triggered.

Environmental

Temperature Range

Operating:	0° to 60° C.
Storage:	-40° to 100° C.

Humidity: 0 to 90% RH, non-condensing.

Power Required

+5 VDC at 180 mA max.
+12VDC at 400 mA max.

Size

9.0 inches long. Requires full-size slot.

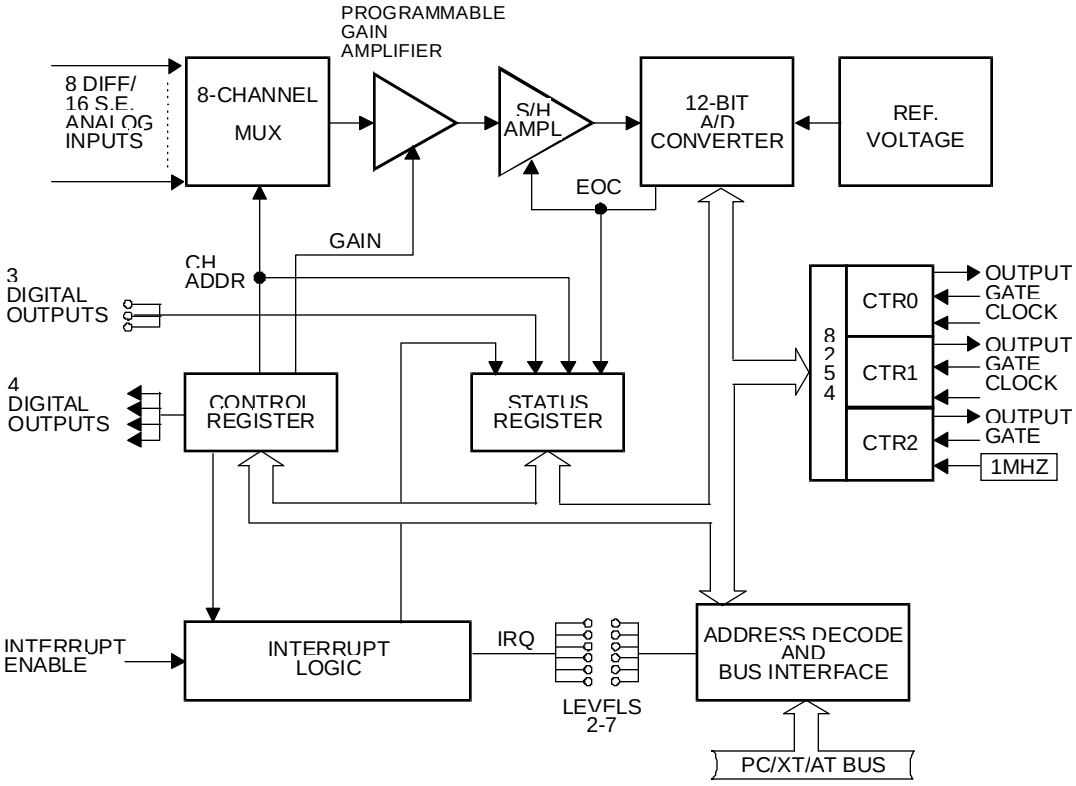


Figure 1-1: AIO8G-P Block Diagram

This page intentionally left blank

Chapter 2: Installation

Software Installation

Installation Program

The software should be installed first before the card is physically installed in the chassis. A setup routine titled **AIO8GSET.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 software is included on the diskette provided with the card. You may make as many back-up copies as required. The software is in a compressed, self-extracting file. Insert the disk in your floppy drive and type:

```
INSTALL.EXE, (or just INSTALL).
```

You will be prompted for the source drive (default is of course the drive with the program disk already installed), and the destination drive\directory. After selection of destination drive, all the programs are extracted and installed in the designated directory. All the driver routines as well as the three sub-directories with sample programs will have been installed.

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 t oassign to the AIO8G-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 AIO8G-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.

Hardware Installation

Before installing the card, be sure to install the software as described above, and run the **AIO8GSET.EXE** program. Check the appropriate section of this manual for further information on address and option selection.

To install the card:

1. Perform the software installation as described above.
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 Jumper as desired.
7. Set the base address.
8. Select differential or single-ended mode for each channel.
9. Install the card in an I/O expansion slot. Attach cable to card.
10. Inspect for proper fit of the card and cable, and tighten screws.
11. Replace the computer cover and apply power.

Base Address Selection

The AIO8G-P cards require eight consecutive address locations in I/O space. Some address locations will be occupied by internal (system) I/O and by other peripheral cards. The base address can be selected to any 8-bit boundary anywhere within the I/O address range 000-3FF hex provided that it does not overlap with other functions. If you are unsure of your available space, run the **FINDBASE** utility provided on the included diskette. Refer to the Findbase section of Chapter 2 for further information.

Hex Range	Usage
000-1FF	Internal System - Not Usable
200-207	Game Control
208-277	Reserved by various manufacturers
278-27F	Parallel Printer (LPT2)
2E8-2EF	Serial Port
2F0-2F7	Reserved
2F8-2FF	Asynchronous Communications (COM2)
300-31F	Prototype Card
320-32F	Hard Disk (XT)
378-37F	Parallel Printer (LPT1)
380-38F	SDLC Communications
3A0-3AF	SDLC Communications
3B0-3BB	MDA
3C0-3CF	EGA
3DO-3DF	CGA
3E0-3E7	Reserved by various manufacturers
3E8-3EF	Serial Port
3F0-3F7	Floppy Disk
3F8-3FF	Asynchronous Comunications (COM1)

The card base address is set by a DIP switch. The DIP switch controls address bits A3 through A9. Lines A2, A1, and A0 are used on the card to select individual registers. How these three lines are used is described later in this manual under PROGRAMMING.

The disk provided with the AIO8G-P card includes an illustrated setup and calibration program. The program is menu-driven and, when you respond to questions asked, shows you how to set switches and jumpers (it is not necessary to have the board installed when you run this program). For base address selection, you are asked what hex-code starting address you desire. When you type that address, the display changes and shows you what the proper DIP switch settings are for that address.

If you prefer to do this yourself instead of following that setup program, first convert the hex address number to binary form. Then, for each "0", set the corresponding switch to ON and for each "1" set the corresponding switch to OFF. The following tables shows likely addresses for the AIO8G-P. An address of hex 300 is recommended.

	A9	**A8**	**A7**	**A6**	**A5**	**A4**	**A3**	**A2**
200H	OFF	ON	ON	ON	ON	ON	ON	ON
210H	OFF	ON	ON	ON	ON	OFF	ON	ON
220H	OFF	ON	ON	ON	OFF	ON	ON	ON
300H	OFF	OFF	ON	ON	ON	ON	ON	ON
310H	OFF	OFF	ON	ON	ON	OFF	ON	ON
320H	OFF	OFF	ON	ON	OFF	ON	ON	ON
350H	OFF	OFF	ON	OFF	ON	OFF	ON	ON

Table 2-1: Address Settings

Address Line	A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0
Binary Representation			1	1	0	0	0	0	0			
Hex Representation	3				0				0			
Switch Settings			OFF	OFF	ON	ON	ON	ON	ON			

Table 2-2: Address Values and Switch Settings

Note: Gray shading in table means these address lines are not available.

ON=0 OFF=1

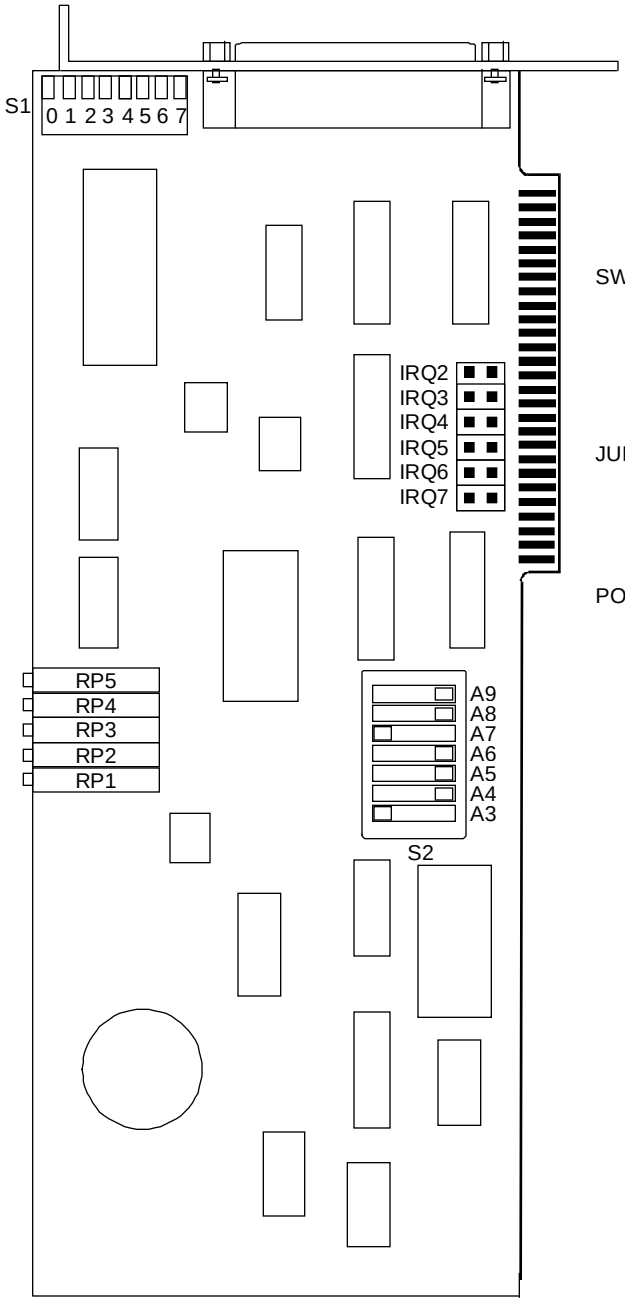
Carefully review the address selection reference table on the preceding page before selecting the card address. If the addresses of two installed functions overlap, you will experience unpredictable computer behavior.

Interrupts

The card supports hardware interrupts IRQ2 through IRQ7. Interrupt level is selected by placing a jumper at the location marked with the level desired. A positive edge triggered interrupt source uses pin 24 to initiate the hardware interrupt. Interrupts may be software enabled or disabled via the INTE bit in the control register.

Differential/Single-Ended Input Selection

AIO8G-P can accept either differential or single-ended analog inputs. DIP switch S1 provides means to select input mode on a channel-by-channel basis. Switch sections 7 through 1 control input selection on channels 0 through 7 respectively. Moving the switch tab to the left, as you look at the rear of the bracket from the back of the chassis, selects single-ended and moving the tab to the right selects differential input mode.



SWITCHES:

S1: Diff./S.E. Selection
S2: Base Address

JUMPERS:

IRQ2-IRQ7: IRQ Level Selection

POTENTIOMETERS:

RP1: Positive Full-Scale Gain Adjust
RP2: Negative Full-Scale Gain Adjust
RP3: Unipolar Offset
RP4: Zero Input Offset Adjust
RP5: Zero Output Offset Adjust

Figure 2-1: Option Selection

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 AIO8G-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 3: Software

Introduction

The diskette provided with the AIO8G-P contains a number of programs. These are the AIO8G-P SETUP program, the AT-16 SETUP program, and the AIO8G-P DRIVER program. In addition, the diskette also contains sample application programs in QuickBASIC, C, and Pascal.

Setup Program

The Setup Program called **AIO8GSET.EXE**, is a bundled program that provide graphics and menus to assist in setting up or configuring and calibrating the cards. The program is menu-driven and includes pictorial presentations. Selection of a menu item results in a presentation on the computer monitor that shows where to install a jumper or how to set up switches. The sections of this program can be performed before the card is installed in the computer (except for calibration).

AT16-P Setup Program

The AT16-P Setup Program, called **SETMUX.EXE** is used when the AIO8G-P is to operate in conjunction with an AT16-P multiplexer expansion card. This program is also menu-driven and includes graphics on the computer monitor to help you program all of the jumpers and switches on the board.

Driver

At the lowest level, the card is programmed using input and output instructions. In BASICA these are the Y = INP (X) and OUT X,Y functions. Assembly language as well as most high level languages have equivalent instructions. To simplify application program development, a device driver program, “AI8GDRV” is included in the software package provided with your card. This driver may be accessed from BASIC by a single CALL statement or linked to QuickBASIC or “C”. The various tasks select the functions of the card, data formatting, error checking, and perform frequently used sequences of instructions. Routines to perform these operations using BASIC INP and OUT statements would require a substantial amount of tedious development (and debug) work and would execute rather slowly. CALLing AI8GDRV saves a lot of programming time.

The following is a description of the driver program. This section describes how to use the driver. It gives an overview of the driver, gives a description of all the tasks, defines all the error codes, and explains how to load the program under BASIC.

The driver views the hardware that it operates on as a 128-channel analog input device. However, the "analog device" can be made up of one or more cards. The basic configuration consists of an AIO8G-P card the eight-channel A/D. To it can be added one or more model

AT16-P 16-channel analog input expansion sub-multiplexer cards and/or LVDT8-P signal conditioning cards. The driver refers to these as the "MUX". The AIO8G-P analog input card is referred to as the "A/D". The combination of eight A/D channels with each channel hooked to a 16-channel MUX provides maximum capability for 128 analog inputs. The driver refers to these 128 inputs as points (or analog points). If no MUXes are attached to the AIO8G-P card, then valid point addresses are only 0, 16, 32, 48, 64, 80, 96, and 112.

The driver also maintains individually assigned gains for each channel for all eight MUXes. This is another reason to refer to the combination of the A/D and the MUX analog input as a point. The gain of the AIO8G-P card alone is programmable AND gain can also be programmed at each MUX channel.

If the programmable gain capability of the AT16-P is desired, the counters must be used to set the gain. Since there are eight gain codes, 0 through 7, outputs of all three counters are used with Counter 0 providing the least significant bit and with Counter 2 providing the most significant bit.

When you use the driver, the details of setting gain are handled by the driver. It's important to note that **IF YOU ARE USING THE COUNTERS TO SET AT16-P GAIN, THEN YOU MUST NOT WRITE ANY VALUES TO THE COUNTER PORTS OR USE TASKS 12, 14, 20, or 21!** Doing so can cause the driver to return inaccurate or invalid values.

To support the point concept, the driver contains two tables. One is a table of MUX and A/D channel address combinations for each point. The other is table of assignable gain codes for each point. All analog input references are done by indexing both tables with the point number, programming a set of control registers to select the actual A/D and MUX channels, and setting the appropriate gain.

This driver also employs another concept that can give your application program a great amount of flexibility. It is called a point list. The driver contains a programmable array of 128 bytes for point addresses. This array is programmable by the application program such as a BASIC program using this driver. The application program has the freedom to assign any combination of points in any order in this array. What this does is allow the application to effectively skip points and scan for data in any order.

The point list is resetable. The initial point on the list can be reset or the whole list can be cleared and, as an option, the list can be set to it's initial state at the time it was loaded. The point list can then be reprogrammed. The initial state of the point list is to address each A/D channel. This initial state assumes that no MUXes are connected and that inputs will be taken in directly through the A/D only. Default point addresses are therefore 0, 16, 32, 48, 64, 80, 96, 112.

Task Descriptions

The call to the driver has the following form:

```
CALL AI8GDRV({task number},{parameters},{status})
```

If you are new to using CALL statements, the following explanation may help you understand how the CALL transfers execution to the driver. Prior to entering the CALL, a DEF SEG = SG statement sets the segment address at which the CALL subroutine has been previously loaded. The three variables within brackets are known as the CALL parameters. Upon CALL execution, the addresses of the variables (pointers) are passed in the sequence written to BASIC's stack. The driver unloads these pointers from the stack and uses them to locate the variables in BASIC's data space so data can be exchanged with them.

Several important format requirements must be met:

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

2. The driver expects parameters to be integer or word 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 statement.

3. The driver will not function properly if arithmetic functions (+, -, x, etc.) are specified within the parameter list brackets of the CALL statement.

4. The driver accepts only variables as parameters. Constants may not be used directly. The values of the variables must be assigned prior to CALLing the driver.

5. Apart from these constraints, you may name the integer variables whatever you wish. Variables should be declared prior to executing the CALL. If this is not done, the simple variables will be declared by default upon execution. Array variables must be dimensioned prior to the CALL for proper operation. Some tasks of the driver require that data be passed in an array. In this case, D%(0) should be specified as the data variable so that the driver can locate the position of the array.

The function name is AI8GDRV. In BASIC, the name of the function must be set to zero before calling it.

The first parameter, {task number}, is a pointer to the integer variable that contains the task number. It is best to use a variable with an explicit integer type (%). Example TASK% or MODE%.

The second parameter, {parameters}, is a pointer to an integer array. This array provides the variable part of the driver function. Some tasks require up to four parameters; some require none. The array must have a minimum size of five items. Example DIM PARAM%(5). As demonstrated it is best to use an array name with the explicit integer type (%).

The third parameter, {status}, is a pointer to an integer variable. The driver will place the return status in this variable. The return status is either a zero or an error code. Again, it is best to use a variable name with an explicit integer type (%). Example STAT% or FLAG%.

The following are examples of BASIC statements to call the AIO8G-P driver.

```
CALL AI8GDRV ( TASK%, PARAM%(1), STAT%)
CALL AI8GDRV ( MODE%, ARGS%(1), ERR%)
CALL AI8GDRV ( MD%, BASADR%, FLAG%)
```

Task Summary

Each of the task descriptions on the following pages will define the required parameter inputs and also the possible return errors.

Task 0 .... Initialize Driver
Task 1 .... Check A/D operations.
Task 2 .... Fetch Gain Code.
Task 3 .... Fetch Point From Point List.
Task 4 .... Assign Gain Code to Range of Points.
Task 5 .... Assign Range of Points to Point List.
Task 6 .... Fetch Immediate Designated Point.
Task 7 .... Fetch Single Assigned Point.
Task 8 .... Fetch Buffered Point Data
Task 9 .... Interrupt-driven Data Acquisition
Task 10 ... Not Used
Task 11 ... Reset.(NOTE: this is multi-function, See Paragraph 3.6.1.13)
Task 12 ... Write Digital Output.
Task 13 ... Read Digital Input.
Task 14 ... Load Counter/Timers.
Task 15 ... Read Counter/Timers.
Task 16 ... Fetch Single Designated Point (High Performance, Buffered).
Task 17 ... Fetch Multiple Points (High Performance, Buffered).
Task 18 ... Set Programmable Gain
Task 19 ... Analog-Triggered Buffered Input
Task 20 ... Frequency Measurement
Task 21 ... Period Measurement

Task 0

This task initializes the driver and must be executed first. If any other task is called before TASK 0, then the driver will return an "invalid task" error message. This task assigns the base address of the card, halts all the counters, programs the interrupt number for interrupt-driven scans, and preassigns the gain code array and the point address list to the eight A/D channels. It also checks as much of the card as possible.

```
task number ......0
1st parameter ....Base I/O address.
2nd parameter ...Interrupt Level
3rd parameter ...Sub-Multiplexer mode
               0= fixed gain at the AT16-P
               1= programmable gain at the AT16-P
```

The base address must be between the addresses 100 hex and 3F8 hex inclusive. Any other addresses will cause an error. This task uses the base address to initialize a set of variables with the appropriate addresses to the AIO8G-P card I/O ports.

e.g. Initialize the driver to base address of hex 2A8,

```
TASK% = 0
PARAM%(1) = &H2A8          'base address of AIO8G-P card
PARAM%(2) = 3'IRQ3
PARAM%(3) = 1              'programmable gain mode
CALL AI8GDRV (TASK%,PARAM%(1),STAT%)
```

Also, the point list is initialized with a list of eight points that correspond to the first channel of each MUX. This in effect prepares the driver to read each channel on the A/D as a default condition.

This task also checks for an operating A/D. If the A/D does not work, an "A/D failed" error code is returned and the driver will not allow any other task until the A/D is operational.

Task 1

This task is a subset of Task 0. It returns an error status if the A/D fails. This function was placed here to check the A/D without having to reset the driver with Task 0.

```
task number ..... 1
1st parameter ... 0
```

e.g. Check the A/D.

```
Task% = 1
CALL AI8GDRV (Task%,PARAM%(1),STAT%)
IF STAT% <> 0 THEN GOTO ....
```

This task requires no parameters, so a dummy parameter must be used.

Task 2

This task returns a gain code. The only parameter it requires is a point address.

```
task number ..... 2
1st parameter ... Point address.
2nd parameter ... Returned gain code.
```

e.g. Fetch gain code from point 5.

```
Task% = 1
PARAM%(1) = 5
PARAM%(2) = 0
CALL AI8GDRV (Task%,PARAM%(1),STAT%)
```

The gain codes are initialized to zero by Task 0. For more detail on gain codes see Task 4.

Task 3

This task returns a point address from the point list. The only parameter required is a point list index.

```
task number ..... 3
1st parameter ... Index to the point list.
2nd parameter ... The returned point number.
```

e.g. Fetch the point address from the eighth entry in the point list.

```
Task% = 3
PARAM%(1) = 8
PARAM%(2) = 0
CALL AI8GDRV (Task%,PARAM%(1),STAT%)
```

The point list is initialized by Task 0. Assignments are made to the point list in Task 5. Task 11 can clear the point list, reset the list index, or reset it to the default list.

NOTE: See Task 0, Task 5 and Task 11 for more details.

Task 4

This task assigns gain codes to the gain table. Software can select gain from 1 to 1000 (gain code 0 to 7) when an AIO8G-P card is used together with an AT16-P analog input expansion sub-multiplexer. If the AIO8G-P card is used alone, the gain is 1 and should be set to gain code 0. Gain codes can be assigned to a maximum of 128 points. A point address may not exceed the value 127. An address exceeding this value will cause an error. Also if the gain code exceeds the value 7, it will cause an error.

The following table lists gains available when the AT16-P is connected:

AT16-P Gain Control			**Gain Code**	**AT16-P Output Range Selection**					
G2	G1	G0		G/1 ON		G/2 OFF	G/1 ON		G/2 OFF
0	0	0	0	Gain	=	1	Gain	=	0.5
0	0	1	1	Gain	=	2	Gain	=	1
0	1	0	2	Gain	=	10	Gain	=	5
0	1	1	3	Gain	=	50	Gain	=	25
1	0	0	4	Gain	=	100	Gain	=	50
1	0	1	5	Gain	=	200	Gain	=	100
1	1	0	6	Gain	=	400	Gain	=	200
1	1	1	7	Gain	=	1000	Gain	=	500

Table 3-1: AT16-P Gain Table

Note: Switch G0, G1, G2 set to ON is equivalent to 0.
Switch G0, G1, G2 set to OFF is equivalent to 1.

The application program assigns gain codes to the table by sending a point range and a gain code for that range. Single gain entries can be assigned by entering a range that begins and ends at the same point. The point range may be entered in opposite order, it makes no difference.

```
task number ..... 4
1st parameter ... Start point address of range.
2nd parameter ... End point address of range.
3rd parameter ... Gain Code.
```

e.g. Assign gain code 3 to point range 8 through 15.

```
Task% = 4
PARAM%(1) = 8
PARAM%(2) = 15
PARAM%(3) = 3
CALL AI8GDRV (Task%,PARAM%(1),STAT%)
```

The point list is initialized by Task 0 to gain code of 0. Task 2 will fetch the assigned gain code.

Note: See Task 0 and Task 2 for more details.

Task 5

This task assigns point numbers to the point list. There are a maximum of 128 entries. If too many entries are made the function will stop assigning entries and return an error code. A point may not exceed the value 127. Assigning a number exceeding this value will cause an error.

The application program assigns points to the list by sending a point range. The points are assigned to the list by expanding a point range into all sets of points within the range. This means, for example, that a range of points from 20 to 30 uses up 11 points in the list.

All points are appended to a previous set of points. Single points can be assigned to the list by entering a range that begins and ends at the same point. This allows the assignment of points in any sequence. Also, a range of points may be entered in opposite order. This allows the scanning of a MUX in reverse order as an example.

```
task number ..... 5
1st parameter ... Start point address of range.
2nd parameter ... End point address of range.
```

e.g. Assign MUX1 and MUX2 to the list in reverse order.

```
Task% = 5
PARAM%(1) = 47
PARAM%(2) = 16
CALL AI8GDRV (Task%,PARAM%(1),STAT%)
```

The point list is initialized by Task 0. Task 3 returns the assignments of the list. Task 11 can clear the point list, reset the list index, or reset it to the default list.

Note: See Task 0, Task 3 and Task 11 for more details.

Task 6

This task fetches an immediate point of data. A point number may not exceed the value 127. A point number exceeding this value will cause an error. The application program sends a point and this function does an immediate access of the A/D and returns the gain code along with the data.

```
task number ..... 6
1st parameter ... Point address.
```

e.g. Fetch a point from A/D channel 2, MUX channel 5.

```
Task% = 6
PARAM%(1) = 37
CALL AI8GDRV (Task%,PARAM%(1),STAT%)
PRINT USING "POINT DATA = ######";PARAM%(2);
```

If the A/D fails an error code is returned.

Task 7

This task fetches a single point of data. The point that is fetched is the next point on the point list. This task requires no parameters. It encodes the point and returns the point number and gain code along with the data. The point number is returned in the first parameter, the data in the second parameter, and the gain code in the third parameter.

Each time the point is fetched from the list, the list index is incremented. The list index can be reset by Task 11. (See Task 11 for more details.)

```
task number ..... 7
```

e.g. Fetch a point of data.

```
Task% = 7
CALL AI8GDRV (Task%,PARAM%(1),STAT%)
PRINT USING "POINT ADRS = ###  ";PARAM(1);
PRINT USING "POINT DATA = ######";PARAM%(2);
PRINT USING "GAIN CODE = #";PARAM%(3)
```

If the A/D fails an error message is returned.

Task 8

This task fetches buffered point data. Task 8 requires three parameters: a data buffer pointer, a point and gain buffer pointer, and a data count. The actual points fetched are determined by the point list.

Note: An error will be returned if the point list is empty (cleared).

Both buffers should be integer buffers of the same length. The count must not exceed the length of the shortest buffer. 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 as well as the data request count.

The data buffer pointer is expected as the first parameter, the point/gain buffer pointer as the second parameter, and the data request count as the third parameter. The actual count is returned in the fourth parameter.

The point and gain for each analog input is returned in the point/gain buffer. The point and gain are packed into one integer with the point number in the upper eight bits and the gain in the lower eight bits.

The list index can be reset by Task 11 before evoking Task 8. (See Task 11 for more details.)

```
task number ..... 8
1st parameter ... data buffer pointer
2nd parameter ... point/gain buffer pointer
3rd parameter ... data count
```

e.g. Fetch 2000 points of data.

```
DIM DATABUF%(2000), POINTBUF%(2000)
Task% = 8
PARAM%(1) = VARPTR(DATABUF%(1))
PARAM%(2) = VARPTR(POINTBUF%(1))
PARAM%(3) = 2000
CALL AI8GDRV (Task%,PARAM%(1),STAT%)
```

Note: If the A/D fails an error message is returned.

Note: See Task 4 for gain code assignments.
See Task 5 for point list assignments.
See Task 11 for resetting the point list.

Task 9

This task provides interrupt-driven data acquisition. The application provides buffer pointers and number of samples. The point address and the gain are transferred with the data.

The AIO8G-P receives interrupts via the INT connector pin on the backplate. If a counter output is to be used, then add a jumper between the counter output pin at the connector to this INT pin. See Sample Program #2 for an example. This task has three sub-tasks.

1. Initiate interrupt scan for {n} data samples
2. Check completion status (End-of-Scan)
3. Transfer data from driver to BASIC

e.g. Fetch 2000 points of data:

```
DIM DATABUF%(2000), PNTBUF%(2000), BUFCNT%
BUFCNT% = {count}
```

Initiate the scan

```
Task% = 9
PARAM%(1) = 1
PARAM%(2) = {# of samples} 'e.g. BUFCNT%
PARAM%(3) = 0 {or buffer offset}   (See Note)
PARAM%(4) = 0 {or buffer segment}  (See Note)
STATUS% = 0
CALL AI8GDRV (Task%,PARAM%(1),STATUS%)
```

Note: Due to incompatibility with some languages, the ability to allocate internal space for buffers is no longer supported. DO NOT PASS ZEROES as the segment count. This allocates buffers that might overwrite system or program data.

Wait for the scan to complete

```
Task% = 9
PARAM%(1) = 2
STATUS% = 0
CALL AI8GDRV (Task%,PARAM%(1),STATUS%)
IF PARAM%(2) <> 0 THEN...  'loop
```

Transfer the data

```
Task% = 9
PARAM%(1) = 3
PARAM%(2) = VARPTR(DATABUF%(1))
PARAM%(3) = VARPTR(PNTBUF%(1))
PARAM%(4) = {samples}      'e.g. BUFCNT%
STATUS% = 0
CALL AI8GDRV(Task%,PARAM%(1),STATUS%)
```

Task 10 - This task is not used in the AIO8G-P at this time.

Task 11

This task does various resets and requires a reset task code. There are five valid reset codes. Any other code will cause an error message.

```
task number ..... 11
1st parameter ... reset code
```

The reset code assignments are as follows:

```
0 ... Reset interrupts
1 ... Reset point list index.
2 ... Clear point list.
3 ... Restore default point list.
4 ... Set the settle-time counter.
```

Reset Code 0 - If the Interrupt mode was set, this sub-task restores the DOS interrupt and the AIO8-P interrupt to the original set up.

```
Task% = 11
PARAM%(1) = 0
CALL AI8GDRV(Task%, PARAM%(1), STAT%)
```

Reset Code 1 - Will set the point list index to zero; i.e.,set index to top of the point list.

```
Task% = 11
PARAM%(1) = 1
CALL AI8GDRV(Task%,PARAM%(1),STAT%)
```

Reset Code 2 - Will clear the point list and set the point list index to zero.

```
Task% = 11
PARAM%(1) = 2
CALL AI8GDRV(Task%,PARAM%(1),STAT%)
```

Reset Code 3 - Will set the point list to the initial default setup and set the index to zero.

```
Task% = 11
PARAM%(1) = 3
CALL AI8GDRV(Task%,PARAM%(1),STAT%)
```

Note: See Task 5 for point list assignments.

Reset Code 4 - Will set the settle-time counter in the driver. This is required for faster computers. For example, a setting of 50 is sufficient for a 25MHz, 386 machine.

```
Task% = 11
PARAM%(1) = 4
PARAM%(2) = 50
```

Task 12

This task writes to the digital output. An eight-bit code is passed in the first parameter but only the four least significant bits carry any meaning. Task 12 does not provide any error messages; any code is valid. If the application sends a code that is greater than four bits, only the lower four bits will be written and the upper bits ignored. DO NOT USE THIS TASK WHEN THE AIO8G-P IS USED WITH AN AT16-P EXPANSION MULTIPLEXER.

```
task number ..... 12
1st parameter ... digital code
```

e.g. Write binary 00000101 to digital output.

```
Task% = 12
PARAM%(1) = &H05(Note, only a hex 5 will be output)
CALL AI8GDRV (Task%,PARAM%(1),STAT%)
```

Task 13

This task reads the digital input. Only the three least significant bits have any meaning. Task 13 requires no parameters and does not provide any error messages. When called, this task returns the three-bit digital input code in the first parameter. When the AIO8G-P is used with an AT16-P, This task will return the multiplexer channel number.

```
task number ..... 13
```

e.g. Read digital input.

```
Task% = 13
CALL AI8GDRV (Task%,PARAM%(1),STAT%)
PRINT HEX$(PARAM%(0))
```

Task 14

This task programs the timer/counter. DO NOT attempt to use this task when the AT16-P multiplexer board is attached and used in the programmable-gain mode. The application program must pass the counter number, the mode the counter should run in, and the 16-bit value for the counter. There are three counters; 0, 1 or 2. Any other number is considered an error. There are six modes; 0 through 5. Any other mode is considered an error. Any load count is valid.

The counter/timer is a type 8254. See Chapter 5 for a description of applications of the counter/timer.

```
task number ..... 14
1st parameter ... counter number
2nd parameter ... counter mode
3rd parameter ... counter load count
```

e.g. Set counter 1 to mode 3 and a count of 5000.

```
Task% = 14
PARAM%(1) = 1
PARAM%(2) = 3
PARAM%(3) = 5000
CALL AI8GDRV (Task%,PARAM%(1),STAT%)
```

Task 15

This task returns the 16-bit integer from one of the timer/counters. There are 3 counters; 0, 1 or 2. Any other number is considered an error.

```
task number ..... 15
1st parameter ... counter number
2nd parameter ... return counter value
```

e.g. Return the value contained in counter 2.

```
Task% = 15
PARAM%(1) = 2
CALL AI8GDRV (Task%,PARAM%(1),STAT%)
PRINT PARAM%(2);"=";HEX$(PARAM%(2))
```

Task 16

This task does high performance buffered acquisition of "n" samples of a single point. Task 16 requires three parameters from the application program; a buffer pointer, the number of samples, and the point number. The point number must be from 0 to 127. Any other point is invalid and will cause an error.

The buffer must be an integer buffer. The count must not exceed the length of the integer buffer. The driver has no criteria to evaluate the validity of the pointer and, thus, it is incumbent upon the application program to supply a valid buffer pointer as well as a valid count.

```
task number ..... 16
1st parameter ... buffer pointer
2nd parameter ... # of samples
3rd parameter ... point number
4th parameter ... {task returns gain code here}
```

e.g. Do 1000 samples of point 17.

```
DIMBUF%(1000)
Task% = 16
PARAM%(1) = VARPTR(BUFFER%(1))
PARAM%(2) = 1000
PARAM%(3) = 17
PARAM%(4) = 0
CALL AI8GDRV (Task%,PARAM%(1),STAT%)
```

Task 17

This task fetches buffered point data. Task 17 requires three parameters: a data buffer pointer, a point buffer pointer, and a data count. The actual points fetched are determined by the point list.

Note: An error will be returned if the point list is empty (cleared).

Both buffers should be integer buffers of the same length. The count must not exceed the length of the shortest buffer. 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 as well as the data request count.

The data buffer pointer is expected as the first parameter, the point buffer pointer as the second parameter, and the data request count as the third parameter. The actual count is returned in the fourth parameter.

The point for each analog input is returned in the point buffer. This task is similar to Task 8 except that it will provide no gain change and it is substantially faster. On a "386" class computer, it will achieve 34000 samples per second.

The list index can be reset by Task 11 before evoking Task 17. (See Task 11 for more details.)

```
task number ..... 17
1st parameter ... data buffer pointer
2nd parameter ... point buffer pointer
3rd parameter ... data count
```

e.g. Fetch 2000 points of data.

```
DIM DATABUF%(2000), POINTBUF%(2000)
Task% = 17
PARAM%(1) = VARPTR(DATABUF%(1))
PARAM%(2) = VARPTR(POINTBUF%(1))
PARAM%(3) = 2000
CALL AI8GDRV (Task%,PARAM%(1),STAT%)
```

NOTE: If the A/D fails an error message is returned.

Task 18

This task configures the programmable-gain amplifier for the desired analog input voltage range. Gain code vs. range is as follows:

Code	**Range**
0	±5V
8	±10V
9	0-10V
10	±0.5V
11	0-1V
12	±0.05V
13	0-0.1V
14	±0.01V
15	0-0.02V

e.g. To select a bipolar 10V range:

```
Task% = 18
PARAM % (1) = 8
CALL AI8GDRV (Task%,PARAM%(1),STAT%)
```

STAT% will return code as follows:

```
0  = no error
1  = base address unknown
16 = invalid gain code
17 = function not available (i.e., AIO8G-P not installed)
```

Task 19

This task provides capability to trigger data acquisition when an input analog voltage exceeds a preset level. Task 19 requires five parameters: a data pointer buffer, the number of conversions to perform, the trigger level, the trigger "side", and the trigger channel.

The data buffer should be an integer array large enough to hold a number of values equal to the number of conversions to perform. The number of conversions can be anywhere within the range 0 to 32767. The trigger level is an integer corresponding to "counts" from the A/D converter. That number can range from 0 to 4095 for unipolar modes and from -2048 to +2047 for bipolar modes. The trigger "side" is either a 1 for high-side triggering or a -1 for low-side triggering. The trigger channel is any valid point in the point list and can range from 0 to 127.

This task continuously monitors the trigger channel and tests the value against the trigger level. On high-side triggering, conversions are started when the channel value exceeds the trigger level. On low-side triggering, conversions are started when the channel value drops below the trigger level. Once the condition for conversions is met, the driver performs a number of acquisitions equal to the value passed by the calling program. Press any key to exit the task before conversions have begun.

```
task number . . . . 19
1st parameter . . . data buffer pointer
2nd parameter . . . number of conversions (0 to 32767)
3rd parameter . . . trigger level(0 to 4096 unipolar, +/-2047 bipolar)
4th parameter . . . trigger side(+1 = high side, -1 = low side)
5th parameter . . . trigger channel (0 to 127)
```

e.g. Perform 200 conversions when a trigger level of 1024 is exceeded on Channel 0:

```
Task% = 19
PARAM%(1) = VARPTR(DATABUF%(1))
PARAM%(2) = 200
PARAM%(3) = 1024
PARAM%(4) = 1
PARAM%(5) = 0
STATUS% = 0
CALL AI8GDRV(Task%,PARAM%(1),STATUS%)
```

The following error codes are possible when calling this task:

```
0  = no error
5  = trigger channel number is out of range
15 = trigger value is invalid
16 = number of conversions is too large
17 = User key press aborted task
```

Task 20

This task provides capability to do frequency measurement with the counter/timer. This task may not be used if the AIO8G-P is being used in conjunction with an AT16-P and gain setting are required at the AT16-P.

Counter 2 is configured to output one-millisecond pulses. This output must be externally connected to the Counter 1 input by means of a jumper at the I/O connector (jumper pin 6 to pin 4). If you set PARAM%(0) = 100, then a 100 millisecond gating pulse is provided at the output of Counter 1. (PARAMETER%(0) can be set to any number <65535 mSec). This output of Counter 1 must be externally connected at the I/O connector to the gate of Counter 0 and to IP2 (jumper pin 5 to pins 21 and 26). The input unknown frequency must be connected between the clock input of Counter 0 (pin 2) and Common (pins 11 or 28).

While the gate signal is low, Counter 0 is set to 65535. Then, when the gate signal goes high, counting is enabled. When the gate signal returns low, PARAM%(1) returns the change in count. This count is proportional to the frequency of the unknown signal and, if gate intervals of 0.1, 1, or 10 seconds are used, the unknown frequency is calculated by simply multiplying PARAM%(1) by 10, 1, or 0.1 respectively.

Task 20 sets the correct counter configurations and performs the entire frequency measurement sequence once the external connections are made. Note that you may still use the output of Counter 1 to trigger interrupts and that this output is in convenient 1-millisecond increments. Also, note that the execution time of Task 20 may be as much as 4*PARAM(0)milliseconds. At higher frequencies, with 10 or 100 mSec gating intervals, the measurement time is brief. But, at lower sampling frequencies (1 or 10 seconds) measurement becomes quite slow. A faster and more accurate method of measuring low frequencies (i.e. <100 Hz), is to measure the period using Task 21 (next section) and derive the frequency from the reciprocal.

Upon entry, initialize the following parameters:

```
Task% = 20
PARAM%(0) = 1000 (gating interval in mSec, 10 to 32767)
PARAM%(1) =  X  (value does not matter)
STATUS%(0) = X  (value does not matter)
CALL AI8GDRV (Task%, PARAM%, STAT%)
```

Upon return, the variables contain data as follows:

```
Task%  = 20
PARAM%(0) = 1000  (unchanged)
PARAM(%1) = data  (count proportional to frequency)
```

The frequency can now be derived from the change in count returned by PARAM%(1) and the gating interval, PARAM%(0) as follows:

```
Freq = PARAM%(1) * 1000 / PARAM%(0)
```

Task 21

This task uses Counter 2 to measure pulse width. This task may not be used if the AIO8G-P is being used in conjunction with an AT16-P and gain settings are required at the AT16-P.

The counter is started by the positive-going edge of the input unknown signal and halted on the negative-going edge. Thus, the half period of a repetitive waveform (or the duration of a positive pulse input) is measured. The maximum period that can be measured is 65.54 milliseconds. Counters 0 and 1 are not used by this task and are free for other uses. The input to be measured must be connected to I/O connector pin 23 (Counter 2 input) and pin 26 (IP2). Task 21 performs all necessary counter initialization and timing operations.

Upon entry, initialize the following parameters:

```
Task% = 21  (task)
PARAM% = X  (value does not matter)
STATUS% = X (value does not matter)
CALL AI8GDRV (Task%, PARAM%, STAT%)
```

Upon return the variables contain data as follows:

```
Task%  = 21  (unchanged)
PARAM% = data (counts decremented)
```

Note that counts greater than 32767 will be returned as negative integers and stored in 2's complement form (BASIC). In other languages, including QuickBASIC, the counts can be up to 65535 as noted in the description of Task 20.

Task 22

This task will start a conversion on a hardware point you specify, using the AIO8G-P's programmable gain feature with the gain code you specify. Although this task uses the Submultiplexer gain code initialized during construction of the Point List in task 4 to program the AIM-16's gain amplifier, the task has been optimized for use without a submultiplexer. To this end, the code will set the on-card gain amplifier to a gain of 1 before changing the A/D channel. Then reset the gain amp. to the specified gain code. By doing this, oversaturation of the sample-and-hold circuitry is avoided when changing from a high-gain channel to a low-gain channel (or vice-versa).

Parameters:

```
Param[0]=point#      /* 0-127, as per Task 6. */
Param[1]=gain code/* 0, 8-15, as per Task 18.  This gain on the AD12-
8G card. */
```

Returns:

```
Params[2]=Counts
Params[3]=Gain Code used on Sub-multiplexer ( if any )
```

Note: This task is similar to a call to Task 18 (to set the AD12-8G gain code) followed by a cal to Task 6 (to convert one input). However, this routine is optimized.

Error Codes

1. Invalid task number: The task number does not fall within the expected range. It also is an invalid task number 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.
4. Point list is full: The point list can only hold 128 entries.
5. Invalid point: The point number 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 7.
7. Invalid reset code: The reset code does not fall within the range of 0 through 4.
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 mode already set: The interrupt mode can only be set up once. A subsequent request has been made to set up the interrupt mode without previously resetting the mode.
11. Invalid interrupt number: The interrupt number does not fall within the range of 2 through 7.
12. Invalid Sub-Task number: Task specified is outside the valid range for a given task.
13. Invalid data buffer: Data buffer is not valid for interrupt- driven data acquisition.
14. Invalid MUX board code: Sub-Multiplexer specified is not 0 or 1.
15. Trigger value is invalid. Must be in the range 0 to 4095 in unipolar mode and -2048 to +2047 in bipolar mode.
16. Number of conversions is too large. Cannot exceed 32,767.
17. User keypress aborted the task.

Chapter 4: Programming

I/O Address Map

AIO8G-P use eight consecutive addresses in I/O space as listed in the following Table:

Address	Read	Write
Base Address + 0	A/D Lo Byte	Start 8 - bit A/D
Base Address + 1	A/D Hi Byte	Start 12 - bit A/D
Base Address + 2	Status Register	Control Register
Base Address + 3	Status & Gain	Gain Control Register
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	----------	Control Counter

Table 4-1: AIO8G-P Address Map

* Base Address + 3 = AIO8G-P only

Control Register

The 8-bit control register is a write-only register. It allows software selection and/or control of functions on the card and associated input expansion cards. At power-up, this register is cleared. This insures that interrupts are disabled, insures that digital outputs OP0-3 are zero, and sets the multiplexer channel address to zero.

Bit Position	D7	D6	D5	D4	D3	D2	D1	D0
Base Address + 2	OP4	OP3	OP2	OP1	INTE	MA2	MA1	MA0

Functions of the bits are as follows:

OP3-OP0 These bits correspond to four general-purpose digital output lines. Bits OP0-OP3 are used for external control functions such as selecting inputs from AT16DC-P analog input expansion cards. Expanding each of the AIO8G-P's eight analog input channels via 16-channel sub-multiplexers on allows a system of up to 128 channels.

INTE A logic high, INTE = 1, enables the interrupt on the card.

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

Status Register

The AIO8G-P Status Register provides card operation information. The Status Register is located at (Base Address + 2), it is a read-only register, and it has the following format:

Bit Position	D7	D6	D5	D4	D3	D2	D1	D0
Base Address + 2	EOC	IP3	IP2	IP1	IRQ	MA2	MA1	MA0

Functions of the bits are as follows:

EOC End of Conversion. If EOC is high (Logic 1) the A/D is busy performing a conversion. Data may be read when EOC is at logic low.

IP3-IP0 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 AIO8G-P card may set the IRQ to logic high (1). It is reset to state 0 by a computer Reset, or a write to the Control Register.

MA2-MA0: These binary-coded bits define the analog multiplexer channel address on the AIO8G-P card (Channels 0 through 7).

Status and Gain Control Register

This is a read/write register located at (Base Address + 3). In the write mode, it is used to set gain at the programmable gain amplifier. In the read mode. it provides multiplexer address and gain data. The data format is:

Bit Position	D7	D6	D5	D4	D3	D2	D1	D0
Write (BASE + 3)	-	-	-	-	R3	R2	R1	R0
Read (BASE + 3)	-	MA2	MA1	MA0	R3	R2	R1	R0

Functions of the bits are as follows:

R3 Through R0		Select the input voltage range		
R3	**R2**	**R1**	**R0**	**Range**
0	0	0	0	±5V
1	0	0	0	±10V
1	0	0	1	0-10V
1	0	1	0	±0.5V
1	0	1	1	0-1V
1	1	0	0	±0.05V
1	1	0	1	0-0.1V
1	1	1	0	±0.01V
1	1	1	1	0-0.02V

MA2-MA0 These binary-coded bits define the analog multiplexer channel address on the AIO8G-P card (Channels 0 through 7).

Reading A/D Data

After the end of a conversion, data from the A/D may be read from registers located at BASE ADDRESS + 0 and BASE ADDRESS + 1. The register located at BASE ADDRESS + 0 contains the least significant four bits from the conversion. The register located at BASE ADDRESS + 1 contains the eight most significant bits from the conversion. If only eight bits resolution is desired, then you only need to read the Hi Byte Register.

LO BYTE Register

Bit Position	D7	D6	D5	D4	D3	D2	D1	D0
Base Address + 0	B9	B10	B11	B12	0	0	0	0

(LSB)

HI BYTE Register

Bit Position	D7	D6	D5	D4	D3	D2	D1	D0
Base Address + 1	B1	B2	B3	B4	B5	B6	B7	B8

(MSB)

The A/D data bits are offset-binary coded. For example, on the ±5V range:

Binary	**Hex**	**Analog Input Voltage**
0000 0000 0000	000	-5.0000V (-Full Scale)
0000 0000 0001	001	-4.9976V
0100 0000 0000	400	-2.5000V (-1/2 Scale)
1000 0000 0000	800	±0V (zero)
1000 0000 0001	801	+0.0024v
1100 0000 0000	C00	+2.5000V (+1/2 Scale)
1111 1111 1111	FFF	+4.9976V (+Full Scale)

Using QuickBasic

The following discussion assumes proficiency with your computer and an understanding of the syntax, statements, and operation of QuickBASIC. References to specific disk drives, subdirectories, paths, etc. are eliminated except for illustrative purposes.

QuickBASIC is a BASIC(A) compiler that provides very close BASIC(A) command support combined with vastly superior execution speed. Creating a QuickBASIC program to use the AIO8G-P driver is done as follows:

1. Locate and place the AI8GDRV.obj file into the same directory as your application.
2. Within your application, dimension the variables used to communicate with the driver.

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

3. Declare the driver as a subroutine.

```
DECLARE SUB AI8GDRV(TASK%, PARAM%, STAT%)
```

4. CALL the driver.

```
CALL AI8GDRV(TASK%, PARAM%(1), STAT%)
```

5. Compile your application to the “obj” form.
6. Link your program to the driver.

```
LINK MYPROG.obj + AI8GDRV.obj, MYPROG;
```

QuickBASIC Incompatibility and Problem Areas

If you are not experienced in using QuickBASIC, be sure to study Appendix A of the “Learningand Using QuickBASIC” manual. It will save a good deal of time and frustration. Also, there are three areas that have been found to be problem areas; reserved words, rounding rules, and overflow errors.

1. Reserved words are a problem in almost any “new” language. For example the array “sub%” will cause an error in QuickBASIC. A good substitute is “mux%.
2. Compilers, in general, use different rounding rules than do interpreters. This is a problem for the value 0.5. Interpreted BASIC(A) will round 0.5 to 1 whereas QuickBASIC will round this to 0.
3. It is often easy to create numbers that are too large to fit into a standard integer variable and, thus create an overflow error. This is especially true when scaling or normalizing data by multiplication. For example:

   X% = A%(I) x 1000/10000

This could easily cause an overflow if A(I) = 20000 because the multiplicand is 20000000 (too large for an integer). QuickBASIC will use default or declared variable types. BASIC(A) will simply adjust its evaluation of the intermediate result. The problem is not the assignment, but the evaluation of the intermediate result. The solution in this case is to use 1000! as the multiplier.

Compiling Programs Outside the QuickBASIC Environment

Application programs may also be compiled outside the QuickBASIC environment by using the following sequence:

```
BC /o MYPROG.BAS
LINK MYPROG.OBJ+AI8GDRV.obj,MYPROG;
```

The "/o" option in the compiler line causes references to the BCOMXX.LIB library (where XX refers to the version of QuicBASIC being used, for example BRUN45.LIB refers to version 4.5) to be placed in the object module so the library response need not be given in the LINK line. This sequence will produce a stand alone executable program that does not require run time support.

As mentioned earlier, an executable program requiring BRUNXX.EXE may also be created:

```
BC MY PROG.BAS;
LINK MY PROG.OBJ+DRIVER.OBJ,MYPROG;
```

Absence of the "/o" option causes references to the BRUN40.LIB library to be placed in the object module so that it need not be given in the link line. The resultant program requires BRUNXX.EXE to be in the root directory \BIN or in the current directory at the time of program execution.

A complete discussion of both stand alone and run time supported programs is given in the "Learning and Using Microsoft QuickBASIC" sections 6.2.2 and 6.2.2.2. Note that the Compiler and Linker expect to find executable modules (.EXE) in the root directory /BIN, and Libraries in the directory named by the environment variable LIB.

Three sample programs are included on the diskette supplied with AIO8G-P. These are presented in QuickBASIC, C, and Pascal. The programs are fully commented.

SAMPLE 1 performs data acquisition on 17 inputs. Sixteen of those inputs are from an AT16DC-P attached to channel 0 of the AIO8G-P and the 17th input is connected to channel 1 of the AIO8G-P. A different gain is used for each of the first eight inputs from the AT16DC-P. The program names are:

```
qsample1.bas   -QuickBASIC program
sample1.c      -"C" program
sample1.pas    -Pascal program
```

SAMPLE 2 demonstrates the use of timer-driven data acquisition on 17 inputs. Sixteen of those inputs are from an AT16DC-P attached to channel 0 of the AIO8G-P and the 17th input is connected to channel 1 of the AIO8G-P. This sample uses the counters to trigger conversions and interrupts to fetch the data. The program names are:

```
qsample2.bas   -QuickBASIC program
sample2.c      -"C" program
sample2.pas    -Pascal program
```

SAMPLE 3 uses the programmable-gain amplifier on the AIO8G-P to scan eight input channels at eight different gain settings. Gain codes 0 and 8 through 14 are used on channels 0 through 7 respectively.

Using Older Versions of BASIC

As mentioned earlier, direct I/O using BASIC INP and OUT statements can be tedious to implement even though a lot of the required programming could be handled in sub-routines. Calling the driver avoids these problems and circumvents some of the execution time delays of interpreted and compiled BASIC and, also, permits interrupt-driven operations which BASIC does not directly support.

BASIC must first load the AIO8G-P driver before it can be used. The BLOAD statement is used to do this. But, before doing a BLOAD, two things must be done. First, the segment where the driver is to be loaded must be defined. Second, the function name (of the driver) must be equated to zero.

Caution must be exercised to avoid loading the driver into memory that is being used by another program. The following information is general and applies to loading any program. It supplements the limited information provided in Appendix C of the IBM BASIC MANUAL.

A BASIC memory segment is 64K bytes long. If BASIC is using the maximum 64K, you will get the following message upon power up or from DOS by entering:

```
A> BASIC(A)
BASICA  The IBM Personal Computer Basic
(DOS X.X)  Version AX.X  Copyright IBM...etc.
60865 Bytes free
Ok
```

When the amount of free memory is less than that shown above for the version of BASIC in use, your computer's memory is fully utilized and BASIC adjusts to this condition by using less than its possible 64K. In this case, you must further contract the BASIC workspace and load the driver at the end of the newly defined workspace. The end of BASIC's data section can be determined with a couple of peek statements. Knowing this, a section of the data space can be allocated for the driver. However, this is not necessary these days with most computers containing 640K bytes of memory or at least memories larger than 256K bytes. This is kind of a left over from the days when the first PC's had only 64K or 128K of memory.

The following fragment is a typical example of loading and executing the first function call using IBM BASIC:

```
150 CLEAR, 49152!
160 SCREEN 0,0,0:KEY OFF:CLS:WIDTH 80
170 DEF SEG = 0
180 SG = 256 * PEEK(&H511) + PEEK(&H510)
190 SG = SG + 49152!/16
200 DEF SEG = SG
210 BLOAD "AI8GDRV.BIN", 0
220 AI8GDRV = 0
230 BASADR% = &H300
240 FLAG% = 0
250 MD% = 0
260 CALL AI8GDRV (MD%, BASADR%, FLAG%)
```

The segment location is set up with the DEF SEG statement. The best thing to do is assign the segment to a high memory location like hex &H5000. The following code fragment is a typical example of loading and executing the first function call using either IBM BASIC or GWBASIC:

```
100 SCREEN 0,0,0:KEY OFF:CLS:WIDTH 80
110 SG = &H5000
220 DEF SEG = SG
230 BLOAD "AI8GDRV.BIN", 0
240 AI8GDRV = 0
250 BASADR% = &H300
260 FLAG% = 0
270 MD% = 0
280 CALL AI8GDRV (MD%, BASADR%, FLAG%)
```

Note: For an in depth description of the form of the CALL driver function (as in statement 280), see the Paragraph 3.4.1.

Calls in Other Languages

To facilitate use of the driver CALL routines in other languages (e.g. FORTRAN, PASCAL, C, etc.), the assembly object code file AI8GDRVC.obj is provided. This was assembled using Borland's Turbo Assembler and may be linked to other object modules generated by "C" compilers etc. The function prototype for entry into the driver is AI8GDRVC.h.

The AI8GDRVC.obj driver may be used by an assembly language routine provided that routine uses the "C" calling conventions. This requires that the parameters be PUSH'ed into the stack in reverse order and that, upon return, your routine must POP six bytes to clear the parameters from the stack. The offset of the three variables is passed.

Note: "C" programs that use the AI8GDRVC.obj driver need to use the Large Memory module.

Another object file, AI8GDRV.obj is provided for use with Pascal and QuickBASIC. It uses the Pascal calling convention.

Note: Both drivers use Far returns requiring Far calls for proper operation.

Using 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 acces to the I/O bus, similar to the C language outport and outportb functions.

InPort, InPortd: Allows read acces to the I/O bus, similar to the C language inport and inportb fucntions.

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

Please refer to the **VBACCES.GBL** file for programming information related to the above functions.

Execution Times

The throughput of the AIO8G-P is a function of many interactions. One limitation is the speed of the A/D conversion. A conversion will take a maximum of 35 microseconds (typically 25 microseconds) and a tightly coded assembly language program, with the A/D operating on one channel, could produce a throughput on the order of 30,000 conversions per second.

However, if you are programming in BASIC, the speed of the interpreter becomes a major limitation. Interpreted BASIC can take a few milliseconds per instruction! For instance, if you are operating TASK 7, a tightly coded BASIC loop may be able to attain speeds of 200 conversions per second. TASK 8 is faster because less time is spent running through the interpreter. Using TASK 8, you can achieve throughput of about 4,000 conversions per second. However, note that since conversions are initiated by software, other interrupt processes may delay conversions.

One way to improve the speed of an interpreted BASIC program is to compile it using a BASIC compiler. If you do this, the 200 conversions per second rate mentioned above can increase to about 3000 conversions per second. Before compiling your program, remove code that BLOADs the driver and all DEF SEG statements that control the location of the routine. These are not required because the linker will locate the AIO8G-P routine in memory automatically. A compiled BASIC program can load the AI8GDRV.bin driver in a manner similar to the interpreted BASIC. The function can then be called with the CALL ABSOLUTE function.

Another way to use the driver with compiled BASIC is to link your compiled BASIC program to the AI8GDRV.obj module provided. You must use DECLARE to define the external procedure as a SUB. Then CALL normally. The QuickBASIC samples provided in Appendix B are examples of how this is done.

The general purpose routines of the AIO8G-P driver include housekeeping overhead such as incrementing the multiplexer, checking the range of input parameters, checking scan limits, etc. Leaner (and thus faster) programs can be written if you only need to cover a specialized requirement. As mentioned earlier, assembly language programs that achieve 20,000 to 30,000 conversions are possible if you pay careful attention to writing the minimum code for specialized requirements.

Chapter 5: Counter/Timer Operations

Summary of Functions

The AIO8G-P contains a type 8254 programmable counter/timer which allows you to implement such functions as generating interrupts and triggering periodic A/D conversions, event counting, digital one-shot, programmable rate generator, square-wave generator, binary rate multiplier, complex wave generator, etc. The 8254 is a flexible and powerful device that consists of three independent, 16-bit, presettable, down counters. Each counter can be programmed to any count between 0 and 65535 in binary format or between 0 and 9999 in BCD format. The maximum clock input frequency is 10 MHz and minimum duty cycle periods are 50 nS high and 50 nS low.

The modes of operation are described in the following paragraphs to familiarize you with the versatility and power of this device. The following definitions apply for use in describing the operation of the 8254:

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

Mode 0: Pulse on terminal count

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

Mode 1: Retriggerable one-shot

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

Mode 2: Rate generator

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

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.

Mode 4: Software triggered strobe

This mode sets the output high when the count is loaded. When gate input is brought high, the counter begins to count down. When the counter reaches zero, the output will go low for one input period. Thus, this mode produces a negative strobe pulse at programmed delay. 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

This is similar to Mode 1 except that the timeout is triggered by the gate input going high and the normally high output going low for one clock period at timeout thus producing a negative strobe pulse. Like Mode 1, the timeout is retriggerable; i.e., a new cycle will commence if the gate input is taken high before a current cycle has timed out.

Programming

On the AIO8G-P the 8254 counters occupy the following addresses:

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

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

D7	D6	D5	D4	D3	D2	D1	D0
SC1	SC0	RW1	RW0	M2	M1	M0	BCD

Definition of bit functions:

SC = Select Counter

SC1	SC0	
0	0	Select Counter 0
0	1	Select Counter 1
1	0	Select Counter 2
1	1	Illegal

M = Mode

M2	M1	M0	
0	0	0	Mode 1
0	0	1	Mode 2
x	1	0	Mode 3
x	1	1	Mode 4
1	0	0	Mode 5
1	0	1	Mode 6

RW = Read/Write

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

BCD

0 = 16 bit binary counter

1 = binary coded decimal, (four codes)

If you attempt to read the counters on the fly, you will possibly get erroneous data; particularly at high input frequencies. This is partially caused by ripple-through of the counter during the read and partially by the fact that the low and high bytes are read sequentially. Thus it is probable that carries will be propagated from the low byte to the high byte during the read cycle. To avoid this problem, you can perform a counter-latch operation just prior to the read cycle. To do this, you load the RW1 and RW0 bits of the Counter Control register with zeros. That latches the count of the selected counter in a 16-bit latch register. The subsequent read operation returns the contents of the latch.

For each counter it is necessary to specify in advance the type of read or write operation to be performed. The choices are writing/reading the high byte, or the low byte, or the low byte followed by the high byte. The latter is the mode most generally used and is specified by setting RW1 and RW0 to ones. (Note: Subsequent write/read operations must be performed in pairs in this sequence. Otherwise, an internal sequencing flip-flop will get out of step.)

Applications

An old adage states that anything that is (or can be made) an analog of time can be easily measured using counting techniques. Six useful applications for the Counter/Timers on the AIO8G-P card are event or pulse counting, frequency output, measuring frequency, measuring pulse width or period, generating time delays, and periodic triggering of the A/D. Of course there are many other applications for counters but these are common uses.

Event Counting

The counter configuration is not of great importance for event or pulse counting because only the countdown capabilities of the counter are used. It can be setup as a square wave generator, retriggerable one-shot, etc. Either Counter 1 or Counter 0 or both of them concatenated can be used. However, Counter 2 cannot be used in this application because it is internally connected to a crystal oscillator clock.

An example program, in BASIC, for event counting is as follows:

```
xxx10        OUTBASADR% + 7,&H30              'setup counter 0 to 'pulse on
                                              terminal 'count, load with 'double
                                              byte 'transfer, count in 'binary
xxx20        OUT BASADR% +4,&HFF              'load low byte with 'FF hex
xxx30        OUT BASADR% +4,&HFF              'load high byte with 'FF hex.
                                              Counter is 'now loaded with the
                                              'maximum count, '65535 (FFFF hex)
xxx40        OUTBASADR% +7,&H00               'latch counter 0
xxx50        XL% = INP (BASADR% +4)           'read low byte
xxx60        XH% = INP (BASADR% +4)           'read high byte
xxx70        CHANGE = 65535 - 256*XH% - XL%   'calculate changes 'in count
```

Frequency Output

Frequency of output is a direct function of the frequency of the clock input and of the count set into the counter. Minimum count (or divisor) is 2 and maximum is 65535. The counters can be concatenated to provide very low frequency out. The clock input to Counter 2 is 1 MHz. If a count of 65535 is set into Counter 2, then the output will be at about 15.3 Hz. If that output is coupled to Counter 1 and Counter 1 is also set for a count of 65535, then the output of Counter 1 will be at about 0.0002 Hz. Further, you could then connect that output to Counter 0 and, if Counter 0 is also set for a count of 65535, then the output would be about 3.55×10^{-9} Hz...about one cycle every 280 million hours!

Measuring Frequency

Frequency is measured by raising the gate input of a counter for a known time interval and counting the number of pulses clocked into the counter during that time. Suitable intervals could be 10, 100, or 1000 milliseconds. (The time interval should be relatively as long as possible in order to minimize quantizing error.)

The gating signal can be derived from Counter 2 and a second cascaded counter, both operating in square wave mode. Also, the computer must be informed at the start and finish of the measurement cycle. To do this, you can monitor that gate input at one of the AIO8G-P digital inputs. Driver Task 20 can be used to measure frequency.

Measuring Pulse Width or Period

Pulse width or half-period of a periodic signal can be measured by applying the signal to be measured to the gate input of a counter and an appropriate frequency input to the counter clock input. During the interval when the gate input is low, load the counter with a full count. When the gate input goes high at the beginning of the measurement, the counter decrements until the gate input goes low at the end of the measurement. Then read the counter and the change in count is a direct function of the gate input signal. Of course the period of a symmetric waveform would be twice that count. If an external frequencysource is not available, Counter 2, which receives a 1 MHz frequency, can be used. Driver Task 21 can be used to measure pulse width or half-period.

Generating Pulse Delay

Accurate time delays can be established by counters. Best time resolution is realized at higher clock frequencies. The retriggerable one-shot mode, Mode 1, is commonly used in this application. In this mode the counter need only be loaded once. As before, when the counter is loaded, the output goes low. When the gate input goes high, the timing delay is initiated and the counter output goes low. If the gate input goes low, counting continues but a new cycle is initiated if the gate input goes high before the time-out delay has expired. At the end of the time-out, the counter reaches zero, the counter output goes high and will remain high until re-triggered by the gate input again going high.

Periodic Triggered of the A/D

A key use of the counter(s) is to provide trigger pulses for continuous periodic A/D conversions. Counter 2 alone can provide sample rates or, for longer periods, you can cascade counters. The output of the counter should be jumpered at the I/O connector to pin 24. When the A/D is triggered by the counter, it can perform conversions continuously in a scan cycle starting and ending at the points set by Task 5 of the driver CALL. If start and end points defined there are equal, then conversions will be performed continuously on that one channel.

This page intentionally left blank

Chapter 6: Calibration and Test

Introduction

Periodic recalibration of AIO8G-P is recommended to maintain best accuracy. The recalibration interval depends to a large extent on the service environment that the card is subjected to. For an environment where there are frequent large changes of temperature and/or where there is vibration, a three-month recalibration interval is recommended. For laboratory or office conditions, six months to one year is acceptable.

A 4 1/2 digit digital multimeter is required as minimum equipment to perform satisfactory calibration. In addition, a voltage calibrator or a stable noise-free DC voltage source that can be used in conjunction with the digital multimeter is required. A card extender such as the Industrial Computer Source EXB21 or EXB21/AT is an inexpensive device that you will find greatly improves accessibility to the card during calibration and will probably be useful with other cards too.

No adjustments are required for the digital I/O or for the Counter/Timers.

Calibrating the A/D

Use the calibration routine in the setup program, AIO8GSET on the diskette provided by Industrial Computer Source to assist you in calibrating the AIO8G-P. AIO8GSET provides visual aids for pre-calibration setup and then provides on-screen instructions for the calibration process.

NOTE: This procedure calls for you to observe a flicker of ±1/2 bit. This will be possible only if no noise is present. A flicker of ±1 bit will cause readings that exceed those allowed. A perfectly valid calibration may still occur since most readings will fall within the ±1/2 LSB calibration range. Adjustment should be made on this basis, with occasional readings which fall outside the range being ignored. No attempt at calibration should be made in noisy locations or with a noisy calibration setup. To calibrate the AIO8G-P card, select the “Calibrate” menu item of the setup program and follow the screen prompts. The procedure is as follows:

1. Install the card in the computer and address it by calibration software.
2. Connect a Screw Terminal Extension card, such as the Industrial Computer Source Model UTB-K, or install a 37-pin female D connector and wire all eight input channels together.
3. Set the calibrator to 0.000V
4. Unipolar Offset Adjust: Set up a gain of x1 and a unipolar range. Connect a multimeter between TP2 and TP3. Adjust RP3 until the multimeter reads 0.000 V.
5. Preamplifier Zero Output Offset Adjust: Keep gain and unipolar setting from step 4. Connect pins 19 and 37 together and connect the multimeter between TP1 and TP3. Adjust RP5 until the multimeter reads 0.000 V.
6. Preamplifier Zero Input Offset Adjust: Change the gain to x500 with the range stillunipolar. Leaving pins 19 and 37 connected together and the multimeter connected across TP1 and TP3, adjust RP4 until the multimeter reads 0.000 V.

7. A/D Negative Full-Scale Adjust: Change the gain back to x1 and select bipolar range. Remove the jumper from pins 19 and 37 of J2. Connect a voltage calibrator negative output to pin 19 and positive output to pin 37. Apply -4.9988 V input and adjust RP2 until the value flickers between -5.0000 and -4.9975 Volts.
8. A/D Positive Full-Scale Adjust: With the same gain, bipolar range, and calibrator connection, apply a +4.9963 V input. Adjust RP1 until the value flickers between +4.9951 and +4.9976 Volts.
9. A/D Converter Mid-Range Check: With the same gain, bipolar range, and calibrator connection, apply a +0.0012V input. Verify that the A/D output flickers between 0.00 and 0.0024V. If a slight error occurs, repeat steps 7 and 8 and adjust for best results.

For maximum accuracy, calibration should be done on the range which is going to be used. A slight change of calibration between ranges may occur due to resistor tolerances inside the 574A A/D chip.

Connector Pin Assignments

Analog and digital I/O signals are connected to the AIO8G-P card via a 37-pin D-type connector that extends through the back of the computer case. The mating connector is an AMP 747304-1 or equivalent. The wiring may be directly from the signal sources or may be on ribbon cable from multiplexer expansion cards or screw terminal accessory boards. Pin assignments are as follows:

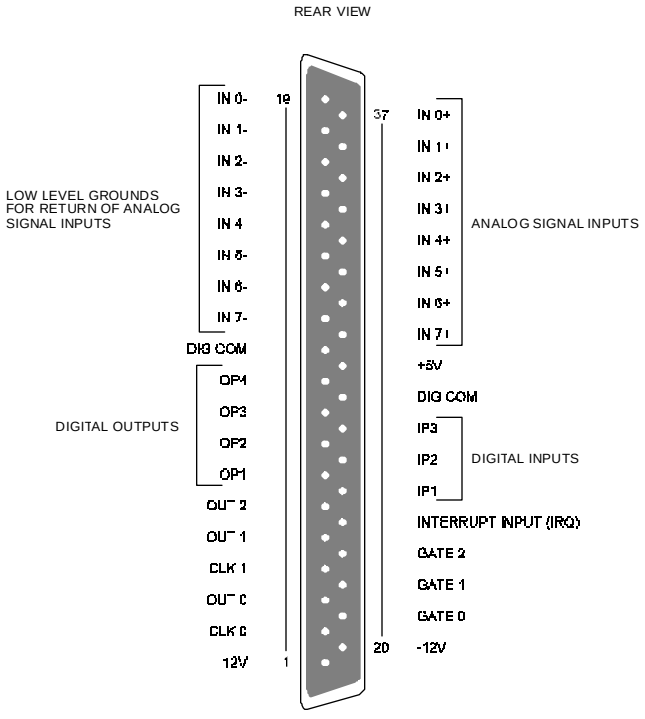


Figure 6-1: Rear View AIO8G-P Connector

Pin	Name	Function
1	+12VDC	+12VDC from P.C bus (output)
2	CLK 0	8254 Counter 0 clock input
3	OUT 0	8254 Counter 0 output
4	CLK 1	9254 Counter 1 clock input
5	OUT 1	8254 Counter 1 output
6	OUT 2	8254 Counter 2 output
7	OP1	Digital output #1, LSB sub-mux channel select
8	OP2	Digital output #2, #2 bit sub-mux channel select
9	OP3	Digital output #3, #3 bit sub-mux channel select
10	OP4	Digital output #4, MSB sub-mux channel select
11	DIG COM	Digtal common. Logic and power supply return
12	IN7-	Analog (low level) grounds
13	IN6-	Analog (low level) grounds
14	IN5-	Analog (low level) grounds
15	IN4-	Analog (low level) grounds
16	IN3-	Analog (low level) grounds
17	IN2-	Analog (low level) grounds
18	IN1-	Analog (low level) grounds

(continued on next page)

Pin	Name	Function
19	IN0-	Analog (low level) grounds
20	-12VDC	-12VDC from P.C bus (output)
21	GATE 0	8254 Counter 0 gate (input)
22	GATE 1	8254 Counter 1 gate (input)
23	GATE 2	8254 Counter 2 gate (input)
24	INT. IN	Interrupt input (pos. edge trigger)
25	IP1	Digital input #1
26	IP2	Digital input #2
27	IP3	Digital input #3
28	DIG. COM	Digital common (same as pin 11)
29	+5VDC	+5VDC from P.C (output)
30	IN7+	Channel #7 analog input
31	IN6+	Channel #6 analog input
32	IN5+	Channel #5 analog input
33	IN4+	Channel #4 analog input
34	IN3+	Channel #3 analog input
35	IN2+	Channel #2 analog input
36	IN1+	Channel #1 analog input
37	IN0+	Channel #0 analog input

Table 6-1: AIO8G-P Pin Assignments

Appendix A: Sample Programs

Five sample programs are included on the diskette supplied with AIO8G-P. These are presented in QuickBASIC, C, and Pascal. The programs are fully commented.

SAMPLE 1 performs data acquisition on 17 inputs. Sixteen of those inputs are from an AT16-P attached to channel 0 of the AIO8G-P and the 17th input is connected to channel 1 of the AIO8G-P. A different gain is used for each of the first eight inputs from the AT16-P. The program names are:

QSAMPLE1.BAS	-QuickBASIC program
SAMPLE1.C	-”C” program
SAMPLE1.PAS	-Pascal program

SAMPLE 2 demonstrates the use of timer-driven data acquisition on 17 inputs. Sixteen of those inputs are from an AT16-P attached to channel 0 of the AIO8G-P and the 17th input is connected to channel 1 of the AIO8G-P. This sample uses the counters to trigger conversions and interrupts to fetch the data. The program names are:

QSAMPLE2.BAS	-QuickBASIC program
SAMPLE2.C	-"C" program
SAMPLE2.PAS	-Pascal program

SAMPLE 3 uses the programmable-gain amplifier on the AIO8G-P to scan eight input channels at eight different gain settings. Gain codes 0 and 8 through 14 are used on channels 0 through 7 respectively.

QSAMPLE3.BAS	QuickBASIC program
SAMPLE3.C	-"C" program
SAMPLE2.PAS	-Pascal program

SAMPLE4 demontrates the use of driver Task 19, Multiple Conversions with Analog Trigger. The program makes four calls to the task to demonstrate high-side and low-side triggering on both bipolar and unipolar ranges.

QSAMPLE4.BAS	QuickBASIC program
SAMPLE4.C	-"C" program
SAMPLE4.PAS	-Pascal program

SAMPLE5 demonstrates the use of driver Task 20, to read the frequency of an unknown TTL waveform. Check program comments for card settings.

QSAMPLE5.BAS	QuickBASIC program
SAMPLE5.C	-"C" program
SAMPLE4.PAS	-Pascal program

This page intentionally left blank

Appendix B: Integer Variable Storage

Data are stored in integer variables (% type) in two's complement form. Each integer variable uses 16 bits or two bytes of memory. Sixteen bits of data is equivalent to values from 0 to 65,535 decimal. However, the two's complement convention interprets the most significant bit as a sign bit so the actual range is -32,768 to +32,768. (A 1 in the MSB position signifies a negative number.)

Integer variables provide the most compact form of storage for 12-bit data from the A/D and 16-bit data from the Counter/Timers. To conserve memory and disk space as well as to optimize execution speeds, all data exchange via the AIO8G-P driver is in integer variables. This may pose a problem when handling unsigned numbers in the range 32,768 to 65,535.

Unsigned integers greater than 32,767 require a signed two's complement format. For example, if you want to load a 16-bit counter with 50,000 decimal, an easy way to determine the binary and/or hex equivalent is to enter BASIC and execute PRINT HEX$(50000). This returns hex C350 and binary 1100 0011 0101 0000. Since the MSB is a 1, this would be stored as a negative integer and, in fact, the correct integer variable value is 50,000 - 65,536 = -15,536. Programming steps to switch between integer and real variables for representation of unsigned numbers between 0 and 65,535 are:

1. From real variable N (0 <= N <= 65535) to integer variable N%:

   ```
   xxx10  IF N<=32767 THEN N%=N ELSE N%=N-65535
   ```

2. From integer variable N% to real variable N:

   ```
   xxx20  IF N%>=0 THEN N%=N ELSE N=N%+65535
   ```

This page intentionally left blank

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.

AIO8G-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: **AIO8G-P**

Manual Revision: **00650-002-35B**

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