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

# MODEL AD12-8G

# USER MANUAL

FILE: MAD12-8G.B3a

## Notice

# Warranty

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

## Terms and Conditions

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

## Coverage

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

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

## Equipment Not Manufactured by ACCES

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

## General

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

# Table of Contents

## List of Figures

## List of Tables

# Chapter 1:  Introduction

The AD12-8G is a multifunction, moderate-speed, analog/digital I/O card with counter/timers. This card may be used with IBM PC/XT/AT and compatible computers. It is 9" long and requires one slot in the computer. All external connections are made through a standard 37-pin D-type connector at the rear of the computer. The following functions are provided by the AD12-8G card.

## Analog Inputs

The card accepts eight single-ended or differential analog inputs. Inputs are DIP switch selectable as either single ended or as differential. This 8-section DIP switch is accessible from outside the computer through an opening in the board mounting bracket and allows channel-by-channel selection of either differential or single-ended mode. The card can withstand a continuous analog input overload of +/-30 volts and brief transients of several hundred volts.

AD12-8G includes a programmable gain amplifier which provides a total of nine software-programmable input voltage 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 AD12-8G card supports up to eight AIM-16, or eight LVDT-8 analog input expansion cards. A 4-bit standard LSTTL logic output from the AD12-8G is used to select one of 16 analog input channels at the AIM-16 and a three-bit logic output is used to select one of eight analog input channels at the LVDT-8. The eight-input multiplexer on the AD12-8G card is software addressable and an input expansion card may be connected to each input, for a maximum of 128 points in the system. To accommodate more than 128 inputs, a second AD12-8G (and companion input expansion cards) can be used.

## Counter/Timer

A type 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. 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 ACCES includes means to use these counters to provide programmable gain commands to the instrumentation amplifier on the AIM-16 expansion multiplexer.

## Interrupts

Interrupts are supported from either external inputs or from end-of-conversion 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:

A. Computer reset signal.

B. Writing a command word to the card. That is either updating the channel selection on the AIM-16 expansion card or on the AD12-8G multiplexer.

## Specifications

### Analog Inputs

| | |
|---|---|
| • No. of Channels: | Eight differential or single-ended inputs (DIP switch selectable). |
| • Voltage Range: | Programmable, +/-5V (default), +/-10V, +/-0.5V, +/-0.01V, 0-10V, 0-1V,0-0.1V, 0-0.02V. |
| • Overvoltage Protection: | +/-30VDC. |
| • Input Impedance: | 10 MΩ or 125 nA at 25 °C. |
| • Multiplexer/Sample and Hold Settle Time: | When gain = 1: 30 sec.<br>When gain = 100: 95 sec. |
| • A/D Trigger Source: | Software selectable: ext. trigger, programmable timer, or program command. |
| • Overall Accuracy: | +/-0.05% of reading +/-1 LSB. |
| • Linearity: | +/-1 LSB. |
| • Resolution: | 12 bit binary. |
| • Temperature Coefficient: | +/-10 V/ °C. zero stability.<br>+/-25 V/ °C. gain stability. |
| • Common Mode Rejection: | 90 db when gain = 1.<br>125 db when gain = 100. |
| • A/D Conversion Time: | 35 sec. maximum, 25 sec. typical. |

## Digital Inputs/Outputs

### Inputs

- Logic Low: 0 to 0.8 V at -0.4 mA source current.
- Logic High: 2.4 to 5.0 V at 20  A 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 (2.5 mA at 0.8 VDC).
- Input Load (Gate and Clock): +/-10 uA, TTL/CMOS compatible.
- Input Clock Frequency: 10 MHz max.
- Active Count Edge: negative edge.
- Clock Pulse Width: 50nS high/50 nS low min.

### Interrupts

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

### Environmental

- Operating Temperature Range: 0 °C. to 60 °C.
- Storage Temperature Range: -40 °C. to 100 °C.
- Humidity: 0 to 90% RH, non-condensing.

- Power Required: +5 VDC at 180 mA. max., +12 VDC at 500 mA. max.
- Size: 9.0 inches long. Requires full-size slot.

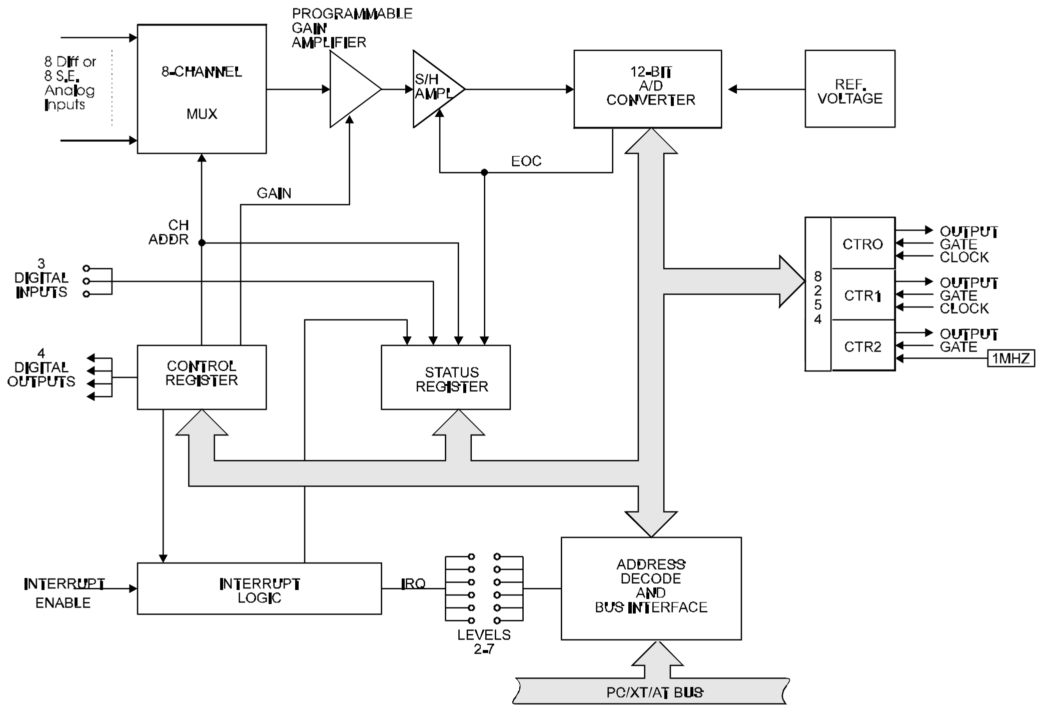


**Figure 1-1:** AD12-8G Block Diagram

## Utility Software

Utility software provided with the AD12-8G includes a menu-driven setup and calibration program, and an AD12-8G device driver in assembly language as well as in binary object code. Two setup programs are provided; one for the AD12-8G itself and one for the AIM-16 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. Also included is a Windows driver for use with VisualBASIC for Windows.

## Enhancements

Capabilities of the AD12-8G can be greatly enhanced by use of one or more of the following hardware devices:

### STA-37 and TAD12-8 Screw Terminal Boards

These terminal boards allow direct input/output signal connections via screw terminals to the I/O lines on the rear connector of the AD12-8G card. The TAD12-8 also provides LED's to show status of digital I/O bits. Both cards provide a breadboard area with +/-12V and +5V computer power. This breadboard area can be used for amplifiers, filters, and other user-assembled circuits.

### AIM-16 Expansion Multiplexer and Instrumentation Amplifier

The AIM-16 is a 16-channel amplifier/multiplexer that features differential-input capability and a choice of either DIP switch selectable gains or software programmable gains. The AIM-16 allows multiplexing of 16 analog input signals to a single AD12-8G analog input channel. As described earlier up to eight AIM-16s can be connected to a single AD12-8G to provide input capability for up to 128 analog inputs.

The AIM-16 includes a low-drift instrumentation amplifier with DIP switch selectable gains of 0.5, 1, 2, 5, 10, 25, 50, 100, 200, 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 AIM-16 or, alternatively, may be jumpered to an unused AIO8G input channel. Open-thermocouple or "break detect" circuitry is provided.

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

### LVDT-8 Multiplexer and Interface Card for LVDTs

This card provides AC excitation and signal conditioning to eight independent LVDT transducers. As many as eight LVDT-8s can be connected to an AD12-8G to accommodate as many as 64 transducers.

# Chapter 2: Installation

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

## CD Installation

### DOS/WIN3.x

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

### WIN95/98/NT

A. Place the CD into your CD-ROM drive.

B. The CD should automatically run the install program after 30 seconds. If the install program does not run, click START | RUN and type d:install, click OK or press K.

C. Follow the on-screen prompts to install the software for this card.

## 3.5-Inch Diskette Installation

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

In a single-drive system, the command is:

```
diskcopy a: a:K
```

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

```
diskcopy a: b:K
```

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

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

a. Place the master diskette into a floppy drive.

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

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

## Directories Created on the Hard Disk

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

### [CARDNAME]

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

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

#### WinRisc.exe

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

### ACCES32

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

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

### SAMPLES

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

## VBACCES

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

## PCI

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

## SOURCE

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

## PCIFind.exe

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

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

## WIN32IRQ

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

### Findbase.exe

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

### Poly.exe

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

### Risc.bat

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

### RISCTerm.exe

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

## Installing the Card

Before carefully read the Option Selection and Address Selection sections of this manual, use the special software program called SETUP provided on CD with the card. That program supplies visual aids to configure the card. Be especially careful with address selection. If the addresses of two installed functions overlap, you will experience unpredictable computer behavior.

### To Install the Card

1. Remove power from the computer.
2. Remove the computer cover.
3. Remove the blank I/O backplate.
4. Set up the base address on the card.
5. Select differential or single-ended input mode for each channel.
6. Install the card in an I/O expansion slot. Make sure that the card mounting bracket is properly screwed into place and that there is a positive chassis ground.
7. Inspect for proper fit of the card and tighten screws.
8. Replace the computer cover and apply power.

To ensure that there is minimum susceptibility to EMI and minimum radiation, it is important that there be a positive chassis ground. Also, proper EMI cabling techniques (cable connect to chassis ground at the mounting bracket, twisted-pair wiring, and, in extreme cases, ferrite level of EMI protection) must be used for input/output wiring.

CE-marked versions of AD12-8G meet the requirements of EN50081-1:1992 (Emissions), EN50082-1:1992 (Immunity), and EN60950: 1992 (Safety).

# Chapter 3:  Option Selection

When reading this section of the manual, refer to the Block Diagram on page 2-2, the Option Selection Map that follows, and the setup program that was provided with your card. Besides Base Address selection, there are only two manual setups to perform; channel-by-channel switch selection for differential or single-ended analog inputs, and interrupt level.

## Analog Input Type

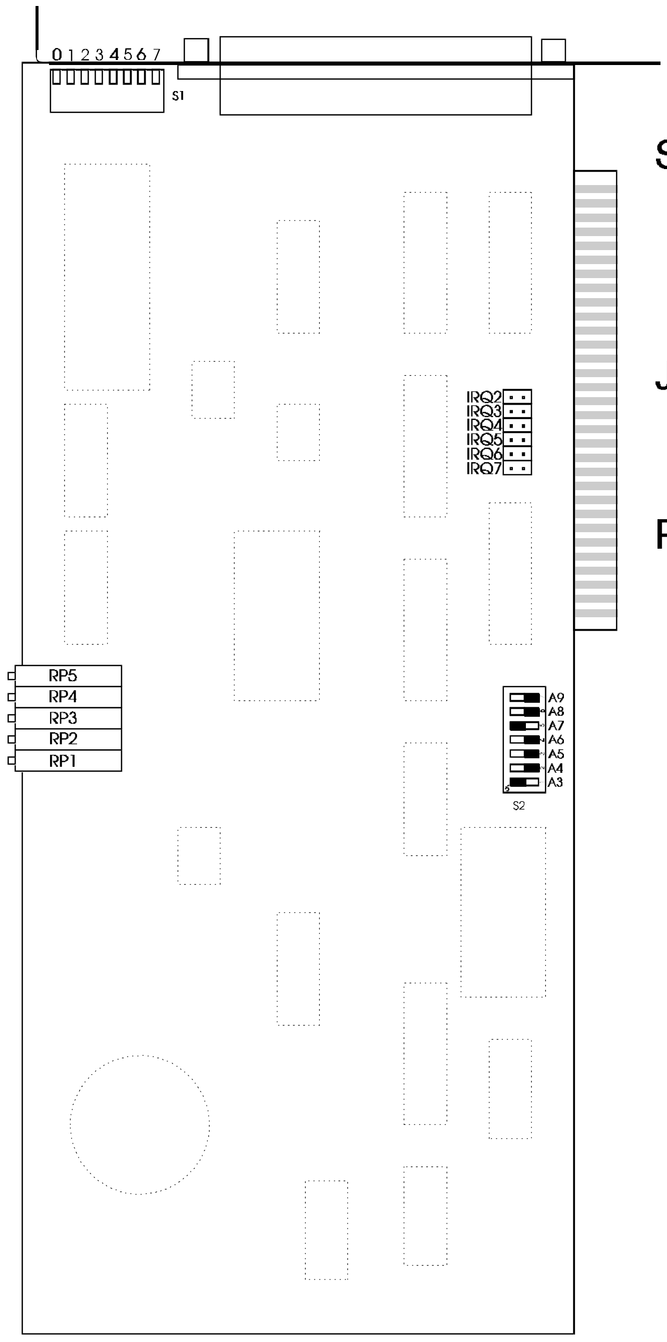
Selection for either differential or single-ended inputs is done using sections of switch S1 located at the upper right rear of the card. Note that S1 is oriented such that, when the card is installed in the computer, it is possible to change settings at the rear of the computer without removing the card from the computer.

### Caution

It is recommended that power be removed before changing the switch settings. Otherwise, damage could result on the card.  Switch sections 7 through 1 control input selection on channels 0 through 7 respectively. Moving the switch tab to the left selects single-ended input and moving the tab to the right selects differential input mode.

## Interrupts

Interrupt commands applied at I/O connector pin 24 are supported. Levels 2 through 7 are available by placing a jumper in one of the locations marked IRQ2 through IRQ7.



## 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 3-1:** Option Selection Map

# Chapter 4:  Address Selection

The AD12-8G card requires 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 100-3FF hex provided that it does not overlap with other functions. The FINDBASE program provided with your card will help you avoid this conflict. If in doubt about what address to use, refer to the following tables for lists of standard address assignments.

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

**Table 4-1:**  Standard Address Assignments for 286/386/486 Computers

The AD12-8G 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 CD provided with the AD12-8G 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, jumpers, etc. 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 how to make 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 example illustrates the process for an address 2D8 hex.

| **Base Address in Hex Code** | 2 | | D | | | | 8 |
|---|---|---|---|---|---|---|---|
| **Conversion Factors** | 2 | 1 | 8 | 4 | 2 | 1 | 8 |
| **Binary Representation** | 1 | 0 | 1 | 1 | 0 | 1 | 1 |
| **Switch Legend** | A9 | A8 | A7 | A6 | A5 | A4 | A3 |
| **Address Line Controlled** | A9 | A8 | A7 | A6 | A5 | A4 | A3 |
| **Address Switch Setup** | OFF | ON | OFF | OFF | ON | OFF | OFF |

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.

# Chapter 5:  Software

## Introduction

The CD  provided by ACCES with the AD12-8G contains a number of programs. These are the AD12-8G Setup and Calibration program, the AIM-16 SETUP program, a windows driver for use with VisualBASIC, and the AD12-8G DRIVER program. In addition, the CD also contains sample application programs in QuickBASIC, C, and Pascal.

## AD12-8G Setup Program

The AD12-8G Setup Program, called "SETMUX.EXE", is a bundled program that provides graphics and menus to assist in setting up or configuring and calibrating the AD12-8G card. It 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).

## AIM-16 Setup Program

The AIM-16 Setup Program, called "SETMUX.EXE", is used when the AD12-8G is to operate in conjunction with an AIM-16 multiplexer expansion card. This program is also menu-driven and includes graphics on the computer monitor to help you program the jumpers and switches on the board.

## AD12-8G Driver

At the lowest level, AD12-8G is programmed using input and output instructions. In BASIC these are the Y = INP(X) and OUT X,Y functions. Assembly language as well as most high level languages have equivalent instructions. Use of these functions usually involves formatting data and dealing with absolute I/O addresses. Although not demanding, this can require many lines of code and necessitates an understanding of the devices, data format, and architecture of the AD12-8G.

To simplify application program development, a device driver program, "A12GDRV" 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, Pascal, or "C". The various tasks select the functions of the AD12-8G 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 A12GDRV saves a lot of programming time.

The following is a description of the driver program and 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 is really made up of one or more cards; an AD12-8G card, and, if used, one or more model AIM-16 analog input expansion sub-multiplexer cards and/or LVDT-8 signal conditioning cards. The driver refers to these as the "MUX". The AD12-8G 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 AD12-8G 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 AD12-8G card alone is programmable AND gain can also be programmed at each MUX channel.

If the programmable gain capability of the AIM-16 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: Counter 0 provides the least significant bit and Counter 2 provides the most significant bit.

When you use the driver, gain-setting details are handled by the driver. It's important to note that if you are using the counters to set AIM-16 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. The application program is free 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 resettable. The initial point on the list can be reset or the 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 re-programmed. 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 A12GDRV({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:

a. 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.
b. 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 unless integer or word variables are used in the CALL statement. (See Appendix C. for more information about integer variable storage.)
c. The driver will not function properly if arithmetic functions (+, -, x, etc.) are specified within the parameter list brackets of the CALL statement.
d. 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.
e. 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 **A12GDRV**. In BASIC, the name of the function must be set to zero before calling it. (See Using the Driver, a later section of this manual.)

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 be dimensioned for 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 A12G driver:

```
CALL A12GDRV ( TASK%, PARAM%(1), STAT%)
CALL A12GDRV ( MODE%, ARGS%(1), ERR%)
```

## 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) |
| 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 Designated Points (High Performance, Buffered). |
| TASK 18 | Set Amplifier Gain. |
| TASK 19 | Analog-Triggered Buffered Input. |
| TASK 20 | Frequency Measurement |
| TASK 21 | Period Measurement |

## Task 0  Initialize Driver

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 pre-assigns 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 AIM-16 |
| | 1= programmable gain at the AIM-16 |

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 AD12-8G card I/O ports.

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

```
TASK% = 0
PARAM%(1) = &H2A8              'base address of AD12-8G card
PARAM%(2) = 3'                 IRQ3
PARAM%(3) = 1                  'programmable gain mode
CALL A12GDRV (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  Check A/D Operation

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 A12GDRV (TASK%,PARAM%(1),STAT%)
IF STAT% <> 0 THEN GOTO ....
```

This task requires no parameters, so a dummy parameter must be used. The parameter does not need to be set.

## Task 2  Fetch Gain Code

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 A12GDRV (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  Fetch Point from Point List

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 A12GDRV (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  Assign Gain Code to Range of Points

This task assigns gain codes to the gain table. Software can select gain from 1 to 1000 (gain code 0 to 7) when an AD12-8G card is used together with an AIM-16 analog input expansion sub-multiplexer. If the AD12-8G 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 AIM-16 is connected:

| **AIM-16 Gain Control** | | | **Gain Code** | **AIM-16 Output Range Selection** | |
|---|---|---|---|---|---|
| **G2** | **G1** | **G0** | | **G/1 ON, G/2 OFF** | **G/1 OFF, G/2 ON** |
| 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 |

**Note:** Switch G0, G1, G2 set to ON is equivalent to 0, 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 A12GDRV (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 Assign Range of Points to Point List

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. Assigning a number, to a point, exceeding 127 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 scanning of a MUX in reverse order if desired.

| | |
|---|---|
| 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 A12GDRV (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  Fetch Immediate Designated Point

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 A12GDRV (TASK%,PARAM%(1),STAT%)
PRINT USING "POINT DATA = ######";PARAM%(2);
```

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

## Task 7  Fetch Single Assigned Point

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 A12GDRV (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  Fetch Buffered Point Data

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 A12GDRV (TASK%,PARAM%(1),STAT%)
```

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 Interrupt-Driven Data Acquisition

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

AD12-8G receives interrupts via the INT connector pin on the backplate. If a counter output is to be used, then add a jumper at the connector between the counter output pin and this INT pin. See Sample Program #2 for an example.

This task has three sub-tasks as follows:

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 A12GDRV (TASK%,PARAM%(1),STATUS%)
```

**Note**

The last two parameters can be used to pass TASK 9 to a data buffer for temporary storage of the data. If 0's are used, then the driver will allocate it's own buffer.

Wait for the scan to complete

```
TASK% = 9
PARAM%(1) = 2
STATUS% = 0
CALL A12GDRV (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 A12GDRV(TASK%,PARAM%(1),STATUS%)
```

## Task 10 Not Used

This task is not used in the AD12-8G at this time.

## Task 11 Reset

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 AD12-8G interrupt to the original set up.

```
TASK% = 11
PARAM%(1) = 0
CALL A12GDRV(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 A12GDRV(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 A12GDRV(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 A12GDRV(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 Write Digital Output

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 AD12-8G is used with an AIM-16 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 A12GDRV (TASK%,PARAM%(1),STAT%)
```

## Task 13 Read Digital Input

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 AD12-8G is being used with an AIM-16 expansion multiplexer, this task will return the multiplexer channel number.

| | |
|---|---|
| task number: | 13 |

e.g.Read digital input.

```
TASK% = 13
CALL A12GDRV (TASK%,PARAM%(1),STAT%)
PRINT HEX$(PARAM%(0))
```

## Task 14 Load Counter/Timers

This task programs the timer/counter. Do not attempt to use this task when the AIM-16 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 Counter Timer Operations for a description of applications of the 8254 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 A12GDRV (TASK%,PARAM%(1),STAT%)
```

## Task 15  Read Counter/Timers

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 A12GDRV (TASK%,PARAM%(1),STAT%)
PRINT PARAM%(2);"=";HEX$(PARAM%(2))
```

## Task 16  Fetch Single Designated Point

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 A12GDRV (TASK%,PARAM%(1),STAT%)
```

## Task 17 Fetch Multiple Designated Points

This task fetches multiple samples of 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 A12GDRV (TASK%,PARAM%(1),STAT%)
```

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

## Task 18 Set Amplifier Gain

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 the bipolar 10V range:

```
TASK% = 18
PARAM% = 8
CALL A12GDRV (TASK%,PARAM%(1),STAT%)
```

STAT% will return code as follows:

0 = no error
6 = invalid gain code

## Task 19 Analog-Triggered Buffered Input

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 positive for high-side triggering or negative 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, or +/-2047 bipolar) |
| 4th parameter: | trigger side (positive value = high side, negative value = 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 A12GDRV(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  Frequency Measurement

This task provides capability to do frequency measurement with the counter/timer. This task may not be used if the AD12-8G is being used in conjunction with an AIM-16 and gain setting is required at the AIM-16.

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 freqiency 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 A12GDRV (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  Period Measurement

This task uses Counter 2 to measure pulse width. This task may not be used if the AD12-8G is being used in conjunction with an AIM-16 and gain setting is required at the AIM-16.

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 A12GDRV (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  Fetch Single Designated Point Using AD12-8G PGA

This task will start a conversion on a hardware point that you specify using the AD12-8G's programmable-gain feature with the gain code that you specify.

Although this task uses the sub-multiplexer gain code initialized during construction of the Point List in Task 4 to program the AIM-16's gain amplifier, the task has ben optimized for use without a sub-multiplexer. To this end, the code will set the on-card gain amplifier to a gain of 1 before changing the A/D channel. And, then, it will reset the gain amplifier to the specified gain code. This prevents saturation of the sample and hold amplifier when changing from a high-gain channel to a low-gain channel or vice versa.

Upon entry, initialize the following parameters:

```
TASK% = 22
PARAM%(0) = Point #     (0 to 127 as per Task 6)
PARAM%(1) = Gain Code (0, 8-15 as per Task 18. This is the gain of the AD12-8G card.)
```

Upon return, the variables contain data as follows:

```
TASK% = 22
PARAM%(2) = Data  (Counts)
PARAM%(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 call to Task 6 (to convert one input).

## Error Codes

| | | |
|---|---|---|
| 1. | Invalid task number: | The task number does not fall within the range of 0 through 21. 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 sub-multiplexer code: | Sub-Multiplexer mode must be defined as either 0 (no programmable gain) or 1 (programmable gain). |
| 15. | Invalid trigger Value: | Must be either 0-4097 in unipolar mode or +/-2047 in bipolar mode. |
| 16. | Invalid number of conversions: | Must not exceed 32,767. |
| 17. | User abort: | An acquisition loop has been terminated by the user at the keyboard. |

## Using the Driver with QuickBASIC

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

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

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

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

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

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

```
DECLARE SUB Ad12gdrv(TASK%, BYVAL PARAM%, STAT%)
```

D. Make your assignment to these variables as desired for the function you wish to perform.

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

```
CALL Ad12gdrv(TASK%, VARPTR(PARAM%(1)), STAT%)
```

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

```
LINK /Q Ad12gdrv.OBJ,Ad12gdrv.QLB,,BQLB45.LIB; [ENTER]
```

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

```
QB /L Ad12gdrv.QLB [ENTER]
```

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

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

```
BC /o YOURPROG;[ENTER]
LINK YOURPROG+Ad12gdrv;[ENTER]
```

## Using the Driver with Basic

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

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

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

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

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

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

```
40      DRIVER = 0
50      BLOAD "ad12gdrv.bin",DRIVER
```

D. Make your assignment to these variables as desired for the function you wish to perform.

E. Make the call to the driver.

```
60      CALL ad12gdrv(TASK%,PARAMS%(1),STATUS%)
```

## Calls in Other Languages

Assembly object code file A12GDRVC.obj is provided to facilitate use of the driver CALL routines in other languages (e.g. FORTRAN, PASCAL, C, etc.). This file 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 A12GDRVC.h.

The A12GDRVC.OBJ driver may also 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 A12GDRVC.OBJ driver must use the Large Memory module

Another object file, A12GDRV.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

## Execution Times

The throughput of the AD12-8G is a function of many interactions. One limitation is the speed of the A/D. 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 BLOAD's the driver and all DEF SEG statements that control the location of the routine. These are not required because the linker will locate the AD12-8G routine in memory automatically. A compiled BASIC program can load the Ad12gdrv.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 Ad12gdrv.obj module provided. You must use DECLARE to define the external procedure as a SUB. Then CALL normally. The QuickBASIC samples provided on the are examples of how this is done.

The general purpose routines of the AD12-8G 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 6:  Programming

## I/O Address Map

The AD12-8G uses eight consecutive addresses in I/O space as listed in the following table:

| Address | Read | Write |
|---|---|---|
| Base Address | 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 Ctl 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 | Unused | Counter Control |

**Table 6-1:** I/O Address Map

## 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, that digital outputs OP0-3 are zero, and that the multiplexer channel address is set to zero.

| **Bit Position** | D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |
|---|---|---|---|---|---|---|---|---|
| **BASE ADDR. +2** | OP3 | OP2 | OP1 | OP0 | 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 AIM-16 analog input expansion cards. Expanding each of the AD12-8G's eight analog input channels via 16-channel sub-multiplexers allows a system of up to 128 channels. |
| INTE: | A logic high, INTE = 1, enables the interrupt on the card. |
| MA2-MA0: | These bits select the analog multiplexer channel address on the card (Channels 0 through 7). |

## Status Register

The AD12-8G 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 ADDR +2** | EOC | IP3 | IP2 | IP1 | IRQ | MA2 | MA1 | MA0 |

Functions of the bits are as follows:

| | |
|---|---|
| EOC: | End of Conversion. Data may be read when EOC is at logic low. If EOC is high (Logic 1) the A/D is busy performing a conversion. |
| IP3-IP1: | 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 AD12-8G 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 bits define the analog multiplexer channel address on the AD12-8G 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 in the Status and Gain Register 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 AD12-8G 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.

To get data, read a 16-bit value (word) from BASE+0, and divide by 16 to eliminate the unused bits. The result is an unsigned integer from 0 to 4096. For bipolar readings, do the same and then subtract 2048.

### LO Byte Register

| **Bit Position** | D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |
|---|---|---|---|---|---|---|---|---|
| **BASE ADDR +0** | B3 | B2 | B1 | B0 | 0 | 0 | 0 | 0 |

(LSB)

### HI Byte Register

| **Bit Position** | D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |
|---|---|---|---|---|---|---|---|---|
| **BASE ADDR +1** | B11 | B10 | B9 | B8 | B7 | B6 | B5 | B4 |

(MSB)

The A/D data bits are binary-coded for unipolar inputs and offset-binary coded for bipolar inputs. For example, the following table shows what the outputs would be 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 (- Half Scale) |
| . . . | . | . |
| . . . | . | . |
| 1000 0000 0000 | 800 | +/- 0V (Zero) |
| 1000 0000 0001 | 801 | +0.0024V |
| . . . | . | . |
| . . . | . | . |
| 1100 0000 0000 | C00 | +2.5000V (+ Half Scale) |
| . . . | . | . |
| . . . | . | . |
| 1111 1111 1111 | FFF | +4.9976V (+ Full Scale) |

# Chapter 7:  Counter/Timer Operations

## Summary of Functions

The AD12-8G contains a type 8254 programmable counter/timer chip 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.

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 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 first 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 except that 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 applications for this mode.

### Mode 4: Software Triggered Strobe

This mode sets the output high when the count is loaded. When the 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 a programmed delay. The counter must be reloaded to repeat the cycle. A low gate input will inhibit the counter.

### Mode 5: Hardware Triggered Strobe

This is similar to Mode 1 except that the timeout is triggered when the gate input goes high. The normally-high output goes 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 AD12-8G 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 the card base address +7. The control word specifies the counter to be programmed, the counter mode, the type of read/write operation, and the modulus.

## Counter Control Register

| 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 | Read Back Command |

**M = Mode**

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

**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 LS Byte and then MS Byte |

**BCD**

0 = 16-bit binary counter
1 = binary coded decimal
(four decades)

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 AD12-8G 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 set up 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     OUT BASADR%+7,&H30          'Set up 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 maximum count, 65535
                                      (FFFF hex)
xxx40     OUT BASADR%+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 count change
```

## 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 3.55x10-9 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 AD12-8G 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 frequency source 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 Delays

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 Triggering 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 the same value, then conversions will be performed continuously on that one channel.

# Chapter 8:  Calibration and Test

## Introduction

Periodic recalibration of AD12-8G 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. No adjustments are required for the digital I/O or for the Counter/Timers.

## Calibrating the A/D

Use the calibration program in the SETUP program provided by ACCES to assist you in calibrating the AD12-8G. SETUP provides visual aids for pre-calibration setup and then provides on-screen instructions for the calibration process.

To calibrate the AD12-8G card, select the "Calibrate A/D" 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 TAD12-8 or STA-37 or install a 37-pin female D type Connector and wire all eight input channels together.
3. Connect a Voltage Calibrator between MUX inputs (+) and the analog ground (-).
4. Set the calibrator to 0.000V.
5. Unipolar Offset Adjust:  Set up a gain of X1 and unipolar range. Connect a multimeter between test points TP2 and TP3. Adjust RP3 until the multimeter reads 0.000V.
6. Amplifier zero-output offset adjust: Keep the X1 gain and unipolar range setup and connect a jumper between pins 19 and 37 on connector J2. Connect the multimeter between test points TP1 and TP3. Adjust RP5 until the multimeter reads 0.000V.
7. Amplifier zero-input offset adjust: Change the gain to X500 and with the range still unipolar. Keep the jumper connected between pins 19 and 37 of connector J2, and the multimeter still connected between test points TP1 and TP3. Adjust RP4 until the multimeter reads 0.000V.

### Note

The next steps apply for A/D converter calibration. For maximum accuracy, A/D converter 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.)

8. A/D converter negative full scale adjust: Change the gain back to X1 and select bipolar range. Remove the jumper between connector J2 pins 19 and 37 and connect a voltage calibrator negative output to pin 19 and positive output to pin 37. Apply a -4.9988V input from the calibrator. Adjust RP2 until the A/D converter output flickers between -5.000 and -4.9975V.

### 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.)

9. A/D converter positive full scale adjust: With the same gain, bipolar range, and calibrator connection, apply a +4.9963V input and adjust RP1 until the A/D converter output flickers between +4.9951 and 4.9976V.
10. 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 8 and 9 and adjust for the best result.

# Chapter 9:  Connector Pin Assignments

Analog and digital I/O signals are connected to the AD12-8G 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 | Name | Function |
|---|---|---|
| 1 | +12VDC | +12VDC from PC Bus (output) |
| 2 | CLK 0 | Counter 0 Clock Input |
| 3 | OUT 0 | Counter 0 Output |
| 4 | CLK 1 | Counter 1 Clock Input |
| 5 | OUT 1 | Counter 1 Output |
| 6 | OUT 2 | Counter 2 Output |
| 7 | OP1 | Digital Output #0 |
| 8 | OP2 | Digital Output #1 |
| 9 | OP3 | Digital Output #2 |
| 10 | OP4 | Digital Output #3 |
| 11 | DIG. COM | Digital Common, Logic and Power Supply Return |
| 12 | INT7- | Channel 7 Analog Input (Low) |
| 13 | INT6- | Channel 6 Analog Input (Low) |
| 14 | INT5- | Channel 5 Analog Input (Low) |
| 15 | INT4- | Channel 4 Analog Input (Low) |
| 16 | INT3- | Channel 3 Analog Input (Low) |
| 17 | INT2- | Channel 2 Analog Input (Low) |
| 18 | INT1- | Channel 1 Analog Input (Low) |
| 19 | INT0- | Channel 0 Analog Input (Low) |
| 20 | -12VDC | -12VDC from PC Bus (Output) |
| 21 | GATE 0 | Counter 0 Gate Input |
| 22 | GATE 1 | Counter 1 Gate Input |
| 23 | GATE 2 | Counter 2 Gate Input |
| 24 | INT. IN | Interrupt Input (pos. edge trig.) |
| 25 | IP1 | Dgital 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 PC (Output) |

| | | |
|---|---|---|
| 30 | INT7+ | Channel #7 Analog Input (High) |
| 31 | INT6+ | Channel #6 Analog Input (High) |
| 32 | INT5+ | Channel #5 Analog Input (High) |
| 33 | INT4+ | Channel #4 Analog Input (High) |
| 34 | INT3+ | Channel #3 Analog Input (High) |
| 35 | INT2+ | Channel #2 Analog Input (High) |
| 36 | INT1+ | Channel #1 Analog Input (High) |
| 37 | INT0+ | Channel #0 Analog Input (High) |

**Table 9-1:** Connector Pin Assignments

# Appendix A:  Sample Programs

Three sample programs are included on the CD supplied with AD12-8G. These are presented in QuickBASIC, C, and Pascal.

SAMPLE 1 performs data acquisition on all eight channels of a standalone AD12-8G. 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 with linear scaling. It also demonstrates the use of a single AIM-16 multiplexer card in conjunction with the AD12-8G. The program names are:

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

SAMPLE 3 demonstrates scan of eight input channels at eight different gains. Gain codes 0 and 8 through 14 are used on channels 0 through 7 respectively. The program names are:

| | |
|---|---|
| qsample3.bas | -QuickBASIC program |
| sample3.c | -"C" program |
| sample3.pas | -Pascal program |

# 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 +32,767. (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 AD12-8G 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 in BASIC. 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:

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

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

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

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

# Appendix C:  Converting BASIC(a) Programs to QuickBASIC Format

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 AD12-8G driver is done as follows:

1. Locate and place the Ad12gdrv.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 Ad12gdrv(TASK%, PARAM%, STAT%)
```

4. CALL the driver.

```
CALL Ad12gdrv(TASK%, PARAM%(1), STAT%)
```

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

```
LINK MYPROG.obj + Ad12gdrv.obj, MYPROG;
```

## QuickBASIC Incompatibility and Problem Areas

If you are not experienced in using QuickBASIC, be sure to study Appendix A of the "Learning and 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+Ad12gdrv.obj,MYPROG;
```

The "/o" option in the compiler line causes references to the BCOM40.LIB library 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 BRUN40.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 BRUN40.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.

## Customer Comments

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

ACCES I/O PRODUCTS, INC.

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