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

# MODEL D/A16-16(8)

# USER MANUAL

FILE: MDA1616.A6a

## Notice

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

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

# 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 D/A16-08 and D/A16-16 are full-size cards that can be installed in any long I/O slot of PC-AT class computers. They contain either 8 or 16 double-buffered digital-to-analog converters (DAC) and provide 8 or 16 independent analog output channels of 16-bit resolution. Each analog output channel can be configured for ranges of:

0V to +5V
0V to +10V
-5V to +5V
-10V to +10V

The Option Selection section of this manual contains a description of how to make these selections.

The analog output channels have a double-buffered input for single-step update and each is addressed at its own I/O location. The DACs have a two-byte (8LSB's+8MSB's) loading structure. The analog outputs can be updated either independently or simultaneously.

Finally, D/A16-08 and D/A16-16 contain automatic reset circuits which reset both D/A outputs to all zeroes at system power-on. Upon power-up or hardware reset, the DAC registers are initialized to a "zero" value and the card is set in the Simultaneous Update mode. The cards also support a unique "Software Clear" capability. This feature permits resetting the DAC output to zero volts without changing the output mode.

Software provided with the card includes setup and calibration programs and some sample programs. The setup and calibration program provides pictorial representation and menu selection on the computer monitor. For setup, of course, it is not necessary that the card be plugged into the computer.

# Specification

## Analog Outputs

- Resolution: 16 Binary bits (0 to 65535 decimal).
- Channels: 16 or 8 Voltage output channels.
- Voltage output ranges at 5mA max.

  0.0 to 5.0 VDC. (76uV/bit)
  0.0 to 10.0 VDC.
  -5.0 to +5.0 VDC.
  -10.0 to +10.0 VDC

## Digital-to-Analog Converter

- AD660 D/A Converter, Double buffered / Simultaneous update.
- Relative Accuracy: ± 0.003%.
- Monotonicity: 15 bits over operating temperature range.
- Settling Time: 8 usec to 0.0008% for full-scale step input.
- Linearity: ± 1 LSB integral non-linearity over rated temperature range.
- Gain Stability: 15 ppm /°C.
- Output Drive Capability: 5mA minimum.
- Short-Circuit Current: 25 mA typical.
- Output Resistance: Less than 0.1 Ω.
- Data Format: 16-bit binary.

## Power Requirements

- +5 VDC at 2.5 A typical (16 channels installed).
- ±15 VDC are developed by internal DC-DC converter.

## Environmental

- Operating Temperature Range: 0 °C. to +60 °C.
- Storage Temperature Range: -20 °C. to +85 °C.
- Humidity: 5% to 95% non-condensing.
- Size: Full size card, 13.3" long (338 mm) by 4.8" high (122 mm).

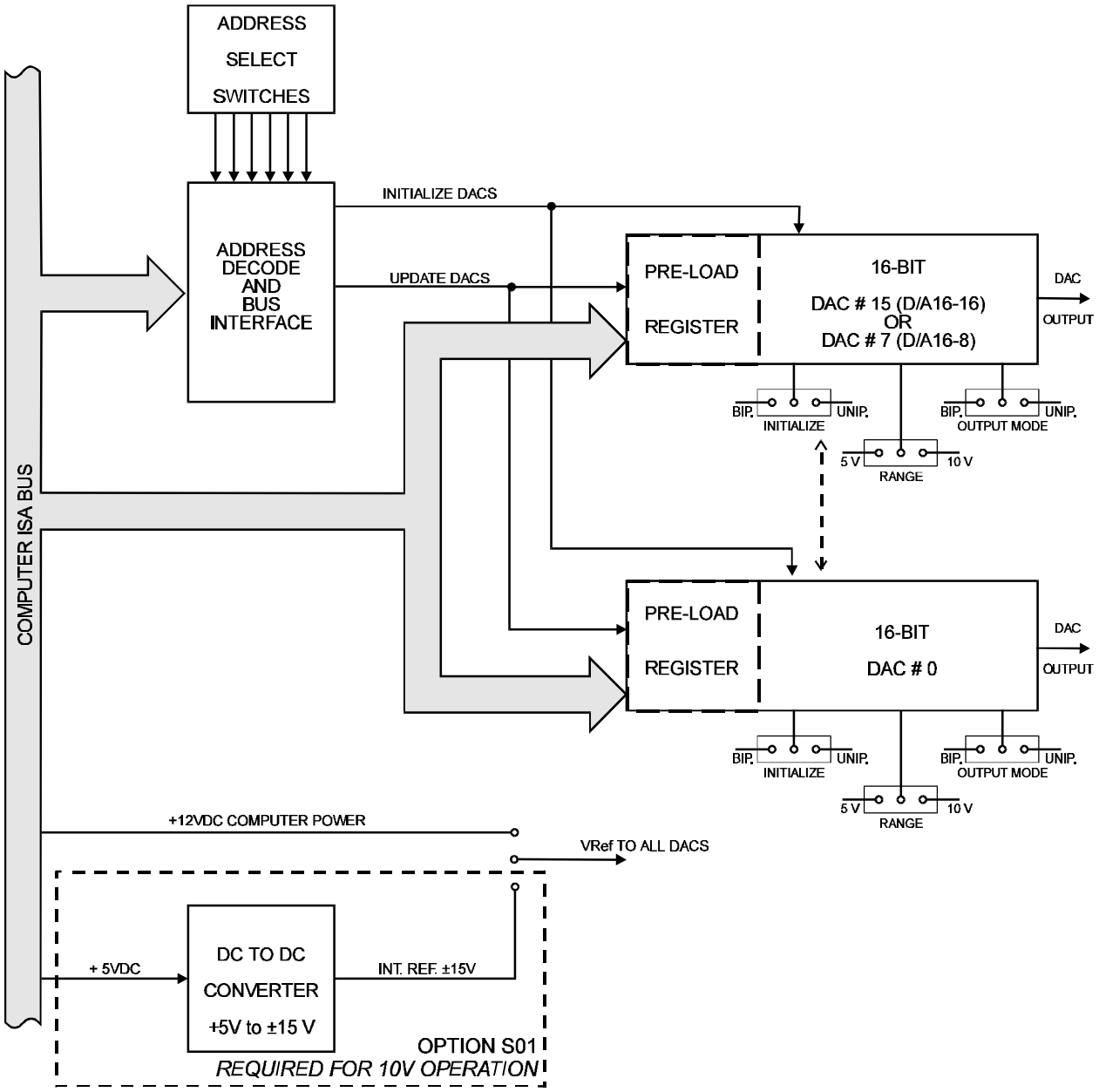



**Figure 1-1:** Block Diagram

# Chapter 2: Installation

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

## CD Installation

### DOS/WIN3.x

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

### WIN95/98/NT

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

## 3.5-Inch Diskette Installation

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

In a single-drive system, the command is:

diskcopy a: a:K

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

diskcopy a: b:K

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

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

a. Place the master diskette into a floppy drive.
b. Change the active drive to the drive that has the diskette installed. For example, if the diskette is in drive A, type a:K.
c. Type installK and follow the on-screen prompts.

## Directories Created on the Hard Disk

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

### [CARDNAME]

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

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

#### WinRisc.exe

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

### ACCES32

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

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

### SAMPLES

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

### VBACCES

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

### PCI

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

### SOURCE

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

### PCIFind.exe

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

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

### WIN32IRQ

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

### Findbase.exe

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

### Poly.exe

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

### Risc.bat

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

### RISCTerm.exe

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

## Installing the Card

Before carefully read the Address Selection and Option Selection sections of this manual and configure the card according to your requirements. Be especially careful with address selection. If the addresses of two installed functions overlap you will experience unpredictable computer behavior. If unsure what locations are available, you can use the FINDBASE program provided on our diskette to locate blocks of available addresses.

### To Install the Card

1. Turn off computer power.
2. Remove the computer cover.
3. Remove the blank I/O backplate.
4. Set switches for selected options. See the option selection section of this manual.
5. Select the base address on the card. See the address selection section of this manual
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. Install the I/O cable.
8. Inspect for proper fit of the card and cables, tighten screws.
9. 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 ground at the I/O connector, twisted-pair wiring, and, in extreme cases, ferrite level of EMI protection) must be used for input/output wiring.

# Chapter 3: Option Selection

Voltage output ranges are determined by jumper placement as described in the following paragraphs. Also, the method to update D/A outputs is programmable as described here and in the Programming section of this manual.

## Output Ranges

To select output voltage ranges (either unipolar or bipolar) set three jumpers located below each DAC output chip. The jumpers select the polarity and range of each DAC channel.

| Voltage Range | Mode "M" and Initialize Jumper "I" | Range Jumper "R" |
|---|---|---|
| 0 to +5 V | Unipolar | 5V |
| -5 to +5 V | Bipolar | 5V |
| 0 to +10 V | Unipolar | 10 V |
| -10 to +10 V | Bipolar | 10 V |

## Analog Outputs Update

Analog outputs are updated under program control in either one of two ways:

a. Each channel is normally updated individually when new data are written to the related high-byte base address. This "individual update" mode may be set by a special read operation as defined in the programming chapter.

-OR-

b. The outputs of all D/A's may be updated simultaneously. This is done by first enabling simultaneous updating for all outputs and then preloading the high and low bytes of each DAC and then initiating a simultaneous update. (Simultaneous update mode is the default on power up.)

Refer to the Programming section of this manual for more detail.

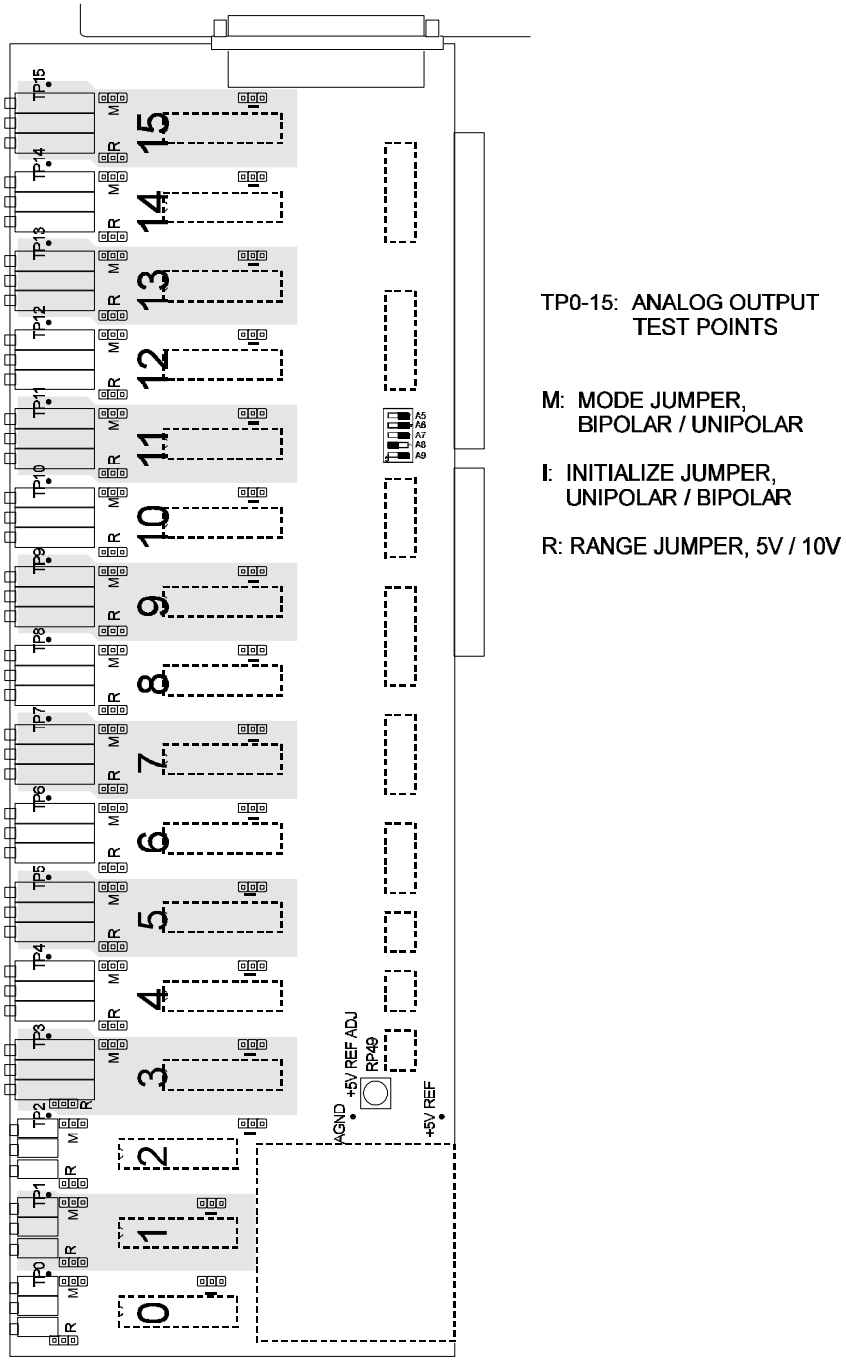



**Figure 3-1:** Option Selection Map

# Chapter 4: Address Selection

The D/A16-08 and D/A16-16 require 16 or 32 consecutive address locations in I/O space, respectively. The starting, or base address, can be selected anywhere within an I/O address range 100-3FF hex (except 1F0 through 1F8) in AT-class computers and 200- 3FF in XT-class computers, providing that the address does not overlap with other functions. If in doubt refer to the table below for a list of standard address assignments. The Base Address Locator program FINDBASE provided will assist you in selecting a base address that will avoid this conflict.

| 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 D/A16-16 and D/A16-08 base address bits A5 through A9 are set by DIP switch S1. The setup program provided with your card includes an interactive base-address selection program. The computer monitor presents a pictorial display of the DIP switch and, when you enter your desired hex base address, the display changes to show proper switch settings for that address.

To understand how this works, consider the following. In order to select the base address, convert the desired address to binary form. Then for each "1" of binary address set the corresponding DIP switch to OFF, and for each "0" of binary address set the corresponding switch to ON.

Here's an example showing how to program the base address to hex 300:

1. Convert hex 300 to binary

   300 (hex) = 11 0000 0000 (binary)

2. Set the Address Selection DIP Switches

The D/A16-08 and D/A16-16 card occupy 16 and 32 bytes of I/O address space, respectively. Address lines A5 through A9 are used to select the base address via DIP switches marked with the same names. Address lines A0 - A4 are used to address registers at the digital-to-analog converters and there are no DIP switches for these five lines.

| **Address** | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
|---|---|---|---|---|---|---|---|---|---|---|
| **Switch** | A9 | A8 | A7 | A6 | A5 | None | | | | |
| **Setting** | OFF | OFF | ON | ON | ON | None | | | | |

# Chapter 5: Programming

Programming the D/A16-16 is very straightforward as there are only two operating modes, three sets of jumpers, and one unique addition. The basic operation of a Digital to Analog card is to write a 16-bit value to a Digital to Analog Converter (DAC) preload register where it is buffered and loaded with an update command to a DAC register which produces the corresponding analog output (Defined by the range and polarity jumpers for that channel).

Upon power-up, or hardware reset, the DAC registers are initialized to a "zero" value and the card is set in Simultaneous Update mode. This ensures that upon power-up the outputs start at zero volts out. (Note: The "I" initialize and "M" polarity mode jumpers should be set identically or the DAC register will be initialized to the incorrect value.) Since the preload register is not cleared upon power-up, but left at an undefined value, a known value must be written to the preload registers before using an update command.

**Simultaneous Update Mode** is the power-up or default mode of operation for the DAC card. When a value is written to a DAC address the output does not change until an output update is commanded via a read to the BASE+8 address. (Alternatively, a read to BASE+10 will update the DAC registers and switch the board to Automatic Update Mode.) While in Simultaneous Update Mode, a single read will load all DAC registers with the value waiting in the preload registers causing all outputs to be updated and changed simultaneously.

**Automatic Update Mode** Changes the DAC output immediately after the new value high-byte is written to the DAC address. If the card is in Simultaneous Update Mode a read to BASE+2 address will change the card back to Automatic Update Mode without updating the outputs. Or, a read to BASE+10 will update all outputs simultaneously and then release the card to the Automatic Update Mode.

**Software Clear** is a unique addition to our DAC card which resets the DAC similar to a hardware reset without changing the operating mode. Just as a hardware reset, the zero output depends on the proper setup of the initialize and polarity mode jumpers (See the power-up paragraph) to produce a zero output. Since the preload registers are not cleared the previous output will be restored from the preload register when the appropriate update command is issued to the DAC channel.

The D/A16-08 and D/A16-16 cards use 16 and 32 consecutive I/O addresses, respectively. The I/O address map is as follows:

| Address | Write * | Read |
|---|---|---|
| Base + 0 | DAC 0 Low Byte | Place card in Simultaneous Mode without updating outputs. |
| Base + 1 | DAC 0 High Byte | |
| Base + 2 | DAC 1 Low Byte | Release card from Simultaneous Mode without updating outputs |
| Base + 3 | DAC 1 High Byte | |
| Base + 4 | DAC 2 Low Byte | |
| Base + 5 | DAC 2 High Byte | |
| Base + 6 | DAC 3 Low Byte | |
| Base + 7 | DAC 3 High Byte | |
| Base + 8 | DAC 4 Low Byte | Update all outputs and place card in Simultaneous Mode |
| Base + 9 | DAC 4 High Byte | |
| Base + 10 | DAC 5 Low Byte | Update all outputs and release card from Simultaneous Mode |
| Base + 11 | DAC 5 High Byte | |
| Base + 12 | DAC 6 Low Byte | |
| Base + 13 | DAC 6 High Byte | |
| Base + 14 | DAC 7 Low Byte | Set all outputs to zero |
| Base + 15 | DAC 7 High Byte | Release zero latch |
| Base + 16 | DAC 8 Low Byte | |
| Base + 17 | DAC 8 High Byte | |
| Base + 18 | DAC 9 Low Byte | |
| Base + 19 | DAC 9 High Byte | |
| Base + 20 | DAC 10 Low Byte | |
| Base + 21 | DAC 10 High Byte | |
| Base + 22 | DAC 11 Low Byte | |
| Base + 23 | DAC 11 High Byte | |
| Base + 24 | DAC 12 Low Byte | |
| Base + 25 | DAC 12 High Byte | |
| Base + 26 | DAC 13 Low Byte | |
| Base + 27 | DAC 13 High Byte | |
| Base + 28 | DAC 14 Low Byte | |
| Base + 29 | DAC 14 High Byte | |
| Base + 30 | DAC 15 Low Byte | |
| Base + 31 | DAC 15 High Byte | |

* Although it is possible to write the low and high bytes separately as shown above, it is much easier to write both bytes with a single OUT DX, AX instruction. In that case, only even addresses are written.

**Table 5-1:** I/O Address Map

## Data Format

| BIT | D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 |
|---|---|---|---|---|---|---|---|---|
| **Low Byte** | B7 | B6 | B5 | B4 | B3 | B2 | B1 | B0 |
| **High Byte** | B15 | B14 | B13 | B12 | B11 | B10 | B9 | B8 |

**For UNIPOLAR ranges:** For Unipolar ranges, data are in true binary form.

```
                    0000 0000 0000 0000 =   ZERO
                    1000 0000 0000 0000 =   1/2 SCALE
                    1111 1111 1111 1111 =   FULL SCALE
                    |                   |
MSB or B15<----|                   |----> B0or LSB
```

**For BIPOLAR ranges:** For Bipolar ranges, data are in offset binary form.

```
                    0000 0000 0000 0000 =   + FULL SCALE
                    1000 0000 0000 0000 =     ZERO
                    1111 1111 1111 1111 =   - FULL SCALE
                    |                   |
MSB or B15  <----|                   |----> B0 or LSB
```

# Chapter 6: Software

The D/A16-08 and D/A16-16 cards are straightforward to program. The following example is in BASIC, but for languages such as C or Pascal the procedure is simplified by their support of two-byte output operations:

To output an analog value with 16-bit resolution, a corresponding decimal number N between 0 and 65536 is calculated (2^12 = 65536).

N/65536 = V(out)/V(full scale)

Then the number is split between high and low bytes, as follows:

```
H = (int)(N / 256);
L = N - (H * 256);
```

Next the data are written to the selected analog output channel. (See the preceding I/O Address Map.) In this example, we will assume analog output on channel zero (AO 0).

```
OUTPORTB (BASE + 0, L);
OUTPORTB (BASE + 1, H);
```

For simplicity, it was assumed that the simultaneous-update capability was not used.

Examples of this routine are found on the sample disk along with examples in other languages.

# Chapter 7: Calibration

Periodic calibration of the D/A16-08 and D/A16-16 cards are recommended if it is used in extreme environmental conditions. The card uses very stable components but vibration, or high-low temperature cycles might result in slight analog output errors.

Factory calibration and periodic calibration of the card includes adjustment of the internal reference voltage unless you are using an external reference voltage.

The suggested sequence for calibration is:

a. Set base address for the card
b. Set range and polarity for each channel
c. Adjust 5V Reference Voltage
d. Adjust Unipolar zero on each channel
e. Adjust Unipolar full scale of each channel
f. Adjust Bipolar zeroes of each channel
g. Check Bipolar negative full scale of each channel
h. Check Bipolar zero of each channel

To calibrate the card, run the setup program and follow the screen prompts. No attempt at calibration should be made in noisy locations or with a noisy calibration setup.

Each DAC output is available between the Analog Ground Pins (located on each end along the top edge of the card) and the Channel Test Point Pins located between each DAC channel's set of calibration potentiometers.

### Note

After changing a channel's voltage range or polarity, the channel may require recalibration for best accuracy.

# Chapter 8: Connector Pin Assignments

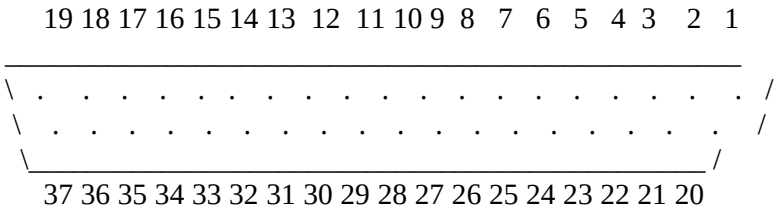
The analog outputs are accessible via a female 37-pin D type connector that extends through the back of the computer case and a DB37P solder cup plug may be used to make connections.

| Pin | Name | Function |
|---|---|---|
| 1 | D/A 0 Out | Analog DAC 0 Output |
| 2 | D/A 1 Out | Analog DAC 1 Output |
| 3 | D/A 2 Out | Analog DAC 2 Output |
| 4 | D/A 3 Out | Analog DAC 3 Output |
| 5 | D/A 4 Out | Analog DAC 4 Output |
| 6 | D/A 5 Out | Analog DAC 5 Output |
| 7 | D/A 6 Out | Analog DAC 6 Output |
| 8 | D/A 7 Out | Analog DAC 7 Output |
| 9 | D/A 8 Out | Analog DAC 8 Output |
| 10 | D/A 9 Out | Analog DAC 9 Output |
| 11 | D/A 10 Out | Analog DAC 10 Output |
| 12 | D/A 11 Out | Analog DAC 11 Output |
| 13 | D/A 12 Out | Analog DAC 12 Output |
| 14 | D/A 13 Out | Analog DAC 13 Output |
| 15 | D/A 14 Out | Analog DAC 14 Output |
| 16 | D/A 15 Out | Analog DAC 15 Output |
| 17 | + 12 Vout | +12 VDC from PC |
| 18 | Analog GND | Analog Ground |
| 19 | -12 Vout | -12 VDC from PC |
| 20 | Return GND | Return Analog Ground |
| 21 | Return GND | Return Analog Ground |
| 22 | Return GND | Return Analog Ground |
| 23 | Return GND | Return Analog Ground |
| 24 | Return GND | Return Analog Ground |
| 26 | Return GND | Return Analog Ground |
| 27 | Return GND | Return Analog Ground |
| 28 | Return GND | Return Analog Ground |
| 29 | Return GND | Return Analog Ground |
| 30 | Return GND | Return Analog Ground |
| 31 | Return GND | Return Analog Ground |
| 32 | Return GND | Return Analog Ground |
| 33 | Return GND | Return Analog Ground |
| 34 | Return GND | Return Analog Ground |
| 35 | Return GND | Return Analog Ground |
| 36 | +5 Vout | +5 VDC from PC |
| 37 | Power GND | Power Ground |

**Table 8-1:** Connector Pin Assignments

## Note

The figure below shows how pins are numbered on D type connectors.



# Appendix A: Sample Programs

Sample programs are provided on the CD with the D/A16-16 or D/A16-08. Sample Program #1 demonstrates general use of the card. This program prompts you for a voltage, calculates the closest actual voltage based on the 16-bit resolution of the DAC, and then programs the card to output this voltage. Sample Program #1 is provided in QuickBASIC, C, and Pascal.

Sample Program #2 will generate a sine, triangle, or sawtooth output waveform. This program is provided in QuickBASIC, C, and Pascal. A sample commented listing of the C language version is as follows (but refer to disk copies for the latest examples):

SAMPLE 2.C

This sample program will generate three different waveforms; sine, triangle, and sawtooth. You have the choice of base address, DAC number, and the number of points per cycle.

The base address entered during program execution should correspond to that set up on the card.

```
#include <math.h>
#include <conio.h>
#include <stdio.h>
#include <dos.h>
#define  PI 3.1415927
unsigned    counts;                 /* number of points per cycle */
unsigned    baseadrs;               /* card base address */
unsigned    dacnum;                 /* DAC used for output */
unsigned    progstruct[20000];      /* buffer to hold points */
```

FUNCTION: setparams() - local routine
PURPOSE: Prompts the user for DAC number, base address, and number of points per cycle.

INPUT : None
CALLS: None
OUTPUT: None

```
void setparms()
{
clrscr();
printf("Enter the base address of your card (in hex)\n");
printf("(Example: 300 : ");
scanf("%x",&baseadrs);
printf("Enter the DAC number you wish to output to (0 or 1):");
```

```
scanf("%u",&dacnum);
dacnum% = 2;
printf("Enter the number of points that you wish to calculate per cycle,\n");
printf("(20000 maximum, program will use modulus if needed);");
scanf("%u",&counts);
counts%=20001;
}                                                    /end setparms*/
```

FUNCTION: sendtoport() - local routine
PURPOSE: Writes point buffer to the DAC until a key is pressed

INPUT: None
CALLS: None
OUTPUT: None

```
void sendtoport()
{
int           i,temp;
long           j;
unsigned char  lowbyte,hibyte;
/*Each point is broken into the high byte and low byte, and then written to the DAC in two     separate
bytes. */
do
   {
   for(i = 0; i <counts,i++)
      {
 temp = progstruct{i} % 256;
      lowbyte = (unsigned char)temp;
      temp = progstruct{i} / 256;
 hibyte = (unsigned char)temp;
 outportb(baseadrs+(dacnum*2),lowbyte);
 outportb(baseadrs+(dacnum*2+1),hibyte);
 }
    }
while (!kbhit());
outportb(baseadrs+(dacnum*2),0);                     /*set DAC to 0 output */
outportb(baseadrs+(dacnum*2+1),0);
}                                                    /*end sendtoport */
```

FUNCTION: sinecurve() - local routine
PURPOSE: Calculate the points to create a sine wave

INPUT: None
CALLS: None
OUTPUT: None

```
void sinecurve()
{
inti;
doublerads,sine;

if (counts == 0) return;                    /*no point -- no curve */

clrscr();
printf("Calculating sine wave points....");

rads = (double) 2 * PI / (counts - 1);      /* rad per count */

for(i = 0;i <counts;i++)
   {
   sine = (sin(rads * i) + 1.0) * 32767;
   progstruct[i] = (unsigned) sine;
   }
clrscr();
printf("Generating sine wave, press any key to stop....");
sendtoport();
}                                           /* end sinecurve */
```

FUNCTION: trianglecurve() - local routine
PURPOSE: Calculate the points to create a trinagle wave

INPUT: None
CALLS: None
OUTPUT: None

```
void trianglecurve(void)
{
inti;
doubleslope,temp;

if (counts == 0) return;                    /* no counts -- no curve */

clrscr();
Printf("Calculating triangle wave points....");
```

```
slope = 65535.0 / counts * 2.0;                    /* waveform slope */
for(i=0;i <counts/2;i++)
   {
   temp = slope * i;
   progstruct[i] = (int)temp;
   temp = 65535 - temp;
   progstruct[i+counts/2+1] = (int)temp;
   }
clrscr();
printf("Generating triangle wave, press any key to stop....");
sendtoport();
}                                                  /* end triangle curve */
```

FUNCTION: sawcurve() - local routine
PURPOSE: Calculate the points to create a sawtooth wave

INPUT: None
CALLS: None
OUTPUT: None

```
void sawcurve()
{
inti;
doubleslope,temp;

if (counts == 0) return;

clrscr();
printf("Calculating sawtooth wave points....");

slope = 4095.0 / counts;                           /* sawtooth slope*/

for(i = 0,i <counts;i++)
   {
   temp = slope * i;
   progstruct[i] = (int) temp;
   }
clrscr();
printf("Generating sawtooth wave, press any key to stop....");
sendtoport();
}                                                  /* end sawcurve */
```

```
FUNCTION: menulist() - local routine
PURPOSE: Display the menu choice on the screen

        INPUT: None
        CALLS: None
        OUTPUT: None
```

```
void menulist(void)
{
clrscr();
printf("\n\n\n");
printf("Your menu selections are:\n");
printf("1. Input Card Data (do this first.)\n"):
printf("2. Sine Curve\n");
printf("3. Triangle Curve\n");
printf("4. Sawtooth Curve\n");
printf("5. End Program, Return to DOS\n");
printf("Input Choice;");
}                                          /* end menulist */
```

```
FUNCTION: main() - local routine
PURPOSE: Controls program execution

        INPUT: None
        CALLS: None
        OUTPUT: None
```

```
void main(void)
{
charmenuchoice;
clrscr();
do
   {
   memset(progstruct, 0, sizeof(int);      /* clear buffer  */
   menulist();                             /* display the menu*/
   menuchoice=getch();                     /* fetch the menu choice */
   switch(menuchoice)                      /* execute menu selection*/
      {
 case '1': setparms();                     /* fetch system parameters*/
  break;
 case '2': sinecurve();                    /* generate a sine wave  */
 break;
 case '3': trianglecurve();                /* generate a triangle wave*/
 break;
 case '4': sawcurve();                     /* generate a sawtooth wave*/
```

```
 break;
 case '5': return;                    /* exit to operating system */
 };
   }
while(1== 1);
}                                              /* end main */
```

# 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