ACCES I/O PRODUCTS INC
10623, Roselle Street, San Diego, CA 92121
Tel. (619)550-9559 FAX (619)550-7322

DIGITAL INPUT/OUTPUT CARD

IOD-72

USER MANUAL

File: MIOD-72.D2

NOTICES

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.

TABLE OF CONTENTS

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 : Enter to change the active drive to the CD-ROM drive.
3. Type I N S T A L L Enter to run the install program.
4. Follow the on-screen prompts to install the software for this card.

WIN95/98/NT

1. Place the CD into your CD-ROM drive.
2. The CD should automatically run the install program after 30 seconds. If the install program does not run, click START | RUN and type D : I N S T A L L, click OK or press Enter.
3. Follow the on-screen prompts to install the software for this card.
4. Click the “Go to ACCES Web” button to check for software updates.

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:

D I S K C O P Y A : A : Enter

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

D I S K C O P Y A : B : Enter

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.

1. Place the master diskette into a floppy drive
2. Change the active drive to the drive that has the diskette installed. For example, if the diskette is in drive A, type A : Enter.
3. Type I N S T A L L Enter 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 ACCES 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 an ACCES 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 address and IRQ 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 n^{th} 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 ACCES 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 installing the card carefully read the ADDRESS SELECTION and OPTION SELECTION Sections of this manual and configure the card according to your requirements. Use the special software program called **SETUP** provided with the card. It supplies visual aids to configure all areas of the board.

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 to locate blocks of available addresses.

To install the card:

1. Remove power from the computer.
2. Remove the computer cover.
3. Remove blank I/O backplate.
4. Install jumpers for selected options. See OPTION SELECTION section of this manual.
5. Select the base address on the card. See ADDRESS SELECTION section of this manual.
6. Loosen the nuts on the strain relief bar and swing top end free.
7. Install the card in an I/O expansion slot. If convenient, select a slot which is adjacent to a vacant slot because this will make cable installation easier.
8. Thread the I/O cables, one by one, through the cutout in the mounting bracket and plug them into the headers.
9. Smooth the cables as close as possible to the card and, while holding them close to the surface of the card, swing the strain relief bar into position and tighten nuts.
10. Inspect for proper fit of the card and cables and tighten screws.
11. Replace the computer cover.

Input/Output cable connections are via two 50-pin headers on the card. A blank mounting bracket is provided with units marked for CE (European) Certification and, for these units, CE-certifiable cable and break-out methodology (cables connect to chassis ground at the aperture, shielded twisted pair wiring, etc.) must be used. Also, it is important that the card bracket be properly screwed in place and that there be a positive chassis ground.

FUNCTIONAL DESCRIPTION

FEATURES

72 Channels of Digital Input/Output.
All 72 I/O Lines Buffered on the Board.
Four and Eight Bit Groups Independently Selectable for I/O.
Hysteresis correction and Pull-Ups on I/O Lines.
Interrupt and Interrupt-Disable Capability.
+5V Supply Available to the User.
Compatible with Industry-Standard I/O Racks like Opto-22, Potter & Brumfield, etc.

APPLICATIONS

Automatic Test Systems.
Robotics
Security Systems, Energy Management.
Relay Monitoring and Control.
Parallel Data Transfer to PC.
Sensing Switch Closures or TTL, DTL, CMOS Logic
Driving Indicator Lights or Recorders

DESCRIPTION

The IOD-72 Board was designed for industrial applications and can be installed in any I/O slot of an IBM PC/XT/AT or compatible computer. Each I/O line is buffered and capable of sourcing 15 mA or sinking 24mA (64 mA on request). The card contains three Programmable Peripheral Interface chips type 8255-5 (PPI) to provide computer interface to 72 I/O lines. Each PPI provides three 8-bit ports A, B, and C. Each 8-bit port can be software configured to function as either inputs or output latches. Port C can also be configured as four inputs and four output latches. The I/O line buffers (74LS245) are configured automatically by hardware logic for input or output use according to direction assignment from a control register in the PPI.

Two I/O lines of each port can be used to interface User Interrupts to the computer. Interrupts are buffered and are enabled by jumper installation or by a combination of jumper installation and a digital input line. You can use Interrupts #2 through #7, #10 through #12, #14 and #15. Interrupts of all ports (one per port) are OR'ed together.

I/O wiring connections are via 50-pin headers on the board. Three flat I/O cables connect IOD-72 to termination panels such as ACCES' model STA-50. Also, this provides compatibility with OPTO-22, Gordos, Potter & Brumfield, etc. module mounting racks. Every second conductor of the flat cables is grounded to minimize the effect of crosstalk

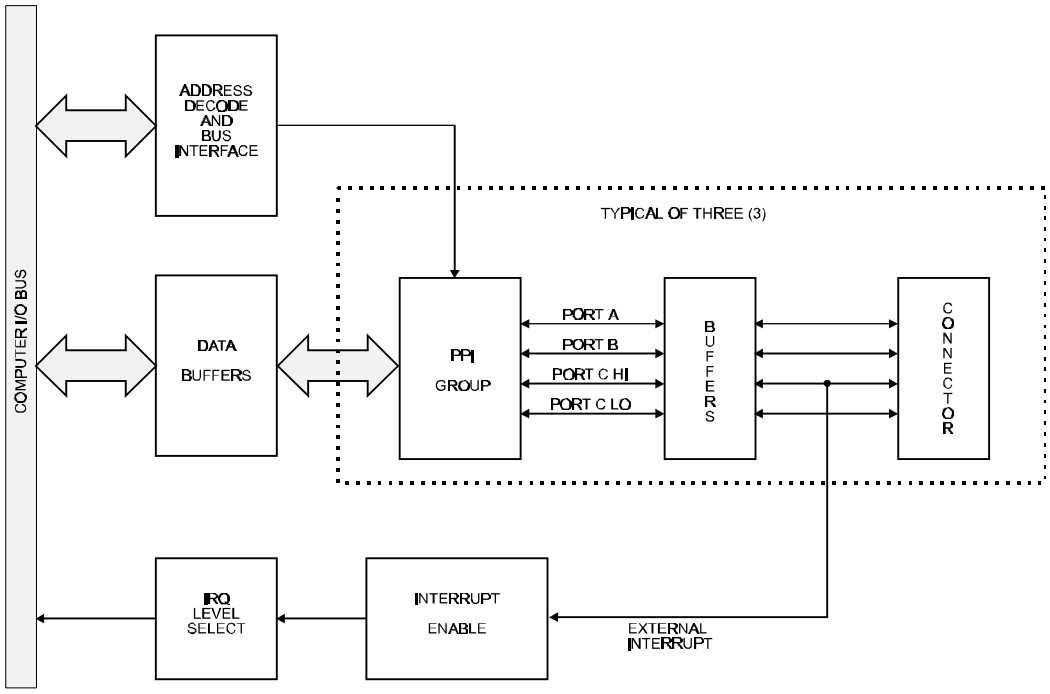
between signals. If needed for external circuits, +5 VDC power is available on each I/O connector pin 49. If you use this power, we recommend that you include a 1A fast-blow fuse in your circuits in order to avoid possible damage to the host computer in the event of a malfunction in those external circuits.

The IOD-72 occupies sixteen bytes of I/O address space. The base address is selectable via a DIP switch anywhere within the range of 000-3FF hex. If in doubt how to select a base address, check your computer Reference Manual. For additional information about setting the base address of IOD-72, see section 4 of this manual.

Utility software provided with the IOD-72 card is an illustrated setup program. Interactive displays show locations and proper settings of DIP switches and jumpers to set up board address, interrupt levels, and interrupt enable. Also, sample programs in Turbo-C and Turbo-Pascal are presented in section 5 of this manual.

IOD-72 BLOCK DIAGRAM

Typical of three sections



OPTION SELECTION

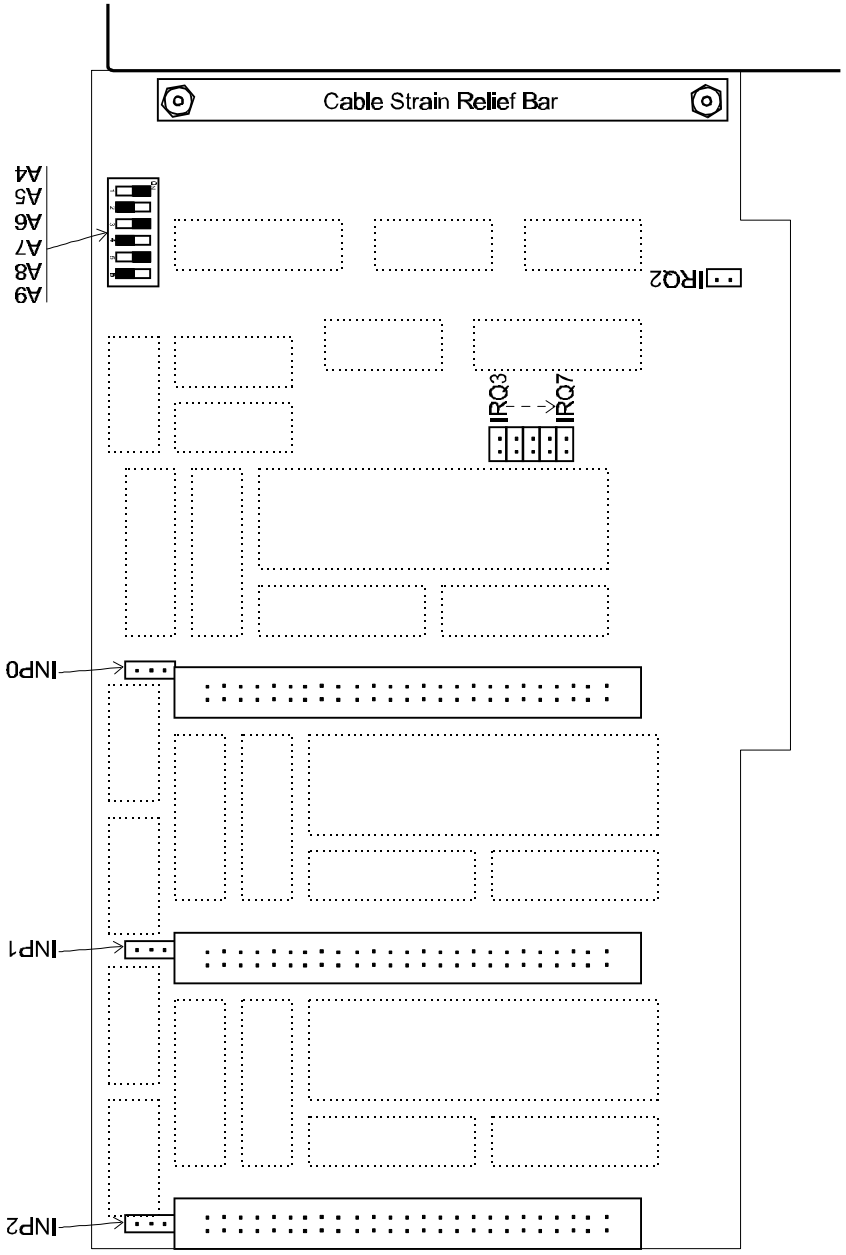
Refer to the setup programs on the CD provided with the card. Also, refer to the IOD-72 Card BLOCK DIAGRAM on the previous pages and the OPTION SELECTION MAP on the following page when reading this section of the manual.

Board Address selection is covered both by the setup program and by section 4 of this manual.

Interrupts are accepted on the I/O connector pin 9 (port C3). The Interrupt signal is positive true. Interrupts are enabled if the IEN jumper is installed or if the IP jumper is installed and the C7 I/O line (connector pin 1) is held low. Interrupts are disabled if neither the IEN or IP jumpers are installed or if the IP jumper is installed but the C7 I/O line is high. Interrupts are directed to levels #2 through #7, #10 through #12, #14, and #15 by jumpers installed at locations labeled IRQ2 through IRQ15.

The foregoing are the only manual setups necessary to use the IOD-72. Input/Output selection is done via software by writing to a control register in each PPI as described in the PROGRAMMING section of this manual.

IOD-72 OPTION SELECTION MAP



ADDRESS SELECTION

The IOD-72 Input/Output Card occupies 16 bytes of I/O space. The card base address can be selected anywhere within an I/O address range 100-3F0 hex in AT's (except 1F0 to 1F8) and 200-3F0 in XT's. However two installed cards cannot share the same address. If in doubt where to assign the base address of the IOD-72, refer to the following tables and the FINDBASE program to find an available address in your system.

STANDARD ADDRESS ASSIGNMENTS FOR PC AND PC/XT COMPUTERS

Hex Range	Usage
000-00F	DMA Chip 8237A-5
020-021	Interrupt 8259A
040-043	Timer 8253-5
060-063	PPI 8255A-5
080-083	DMA Page Register
0AX	NMI Mask Register
0CX	Reserved
0EX	Reserved
100-1FF	Not usable
200-20F	Game Control
210-217	Expansion Unit
220-24F	Reserved
278-27F	Reserved
2F0-2F7	Reserved
2F8-2FF	Asynchronous Comm'n (Secondary)
300-31F	Prototype Card
320-32F	Fixed Disk
378-37F	Printer
380-38C**	SDLC Communications
380-389**	Binary Synchronous Comm. (Secondary)
3A0-3A9	Binary Synchronous Comm. (Primary)
3B0-3BF	IBM Monochrome Display/Printer
3C0-3CF	Reserved
3D0-3DF	Color/Graphics
3E0-3E7	Reserved
3F0-3F7	Diskette
3F8-3FF	Asynchronous Comm'n. (Primary)

** These options can not be used together - addresses overlap

STANDARD ADDRESS ASSIGNMENTS FOR 286/386/486 COMPUTERS

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)

To set a desired board address , refer to the illustrated Board Address setup program provided with the card. Type the desired address in hexadecimal code and the graphic display shows you how to set the ADDRESS SETUP switches. These switches are marked A4-A9 and form a binary representation of the address in negative-true logic. (Assign '0' to all ADDRESS SETUP switches turned ON, and assign '1' to all ADDRESS SETUP switches turned OFF.)

Switch Identification	A9	A8	A7	A6	A5	A4
Address Line Controlled	A9	A8	A7	A6	A5	A4

The following example illustrates switch selection corresponding to hex 2D0 (or binary 10 1101 xxxx). The "xxxx" represents address lines A3, A2, A1, and A0 used on the Card to select individual registers at the PPI's. See Section 5, PROGRAMMING.

Hex Representation	2		D			
Conversion Multipliers	2	1	8	4	2	1
Binary Representation	1	0	1	1	0	1
Setup	OFF	ON	OFF	OFF	ON	OFF
Switch ID	A9	A8	A7	A6	A5	A4

CAUTION

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

SOFTWARE

ACCES supplies several programs to support the IOD-72 Digital I/O card and, also, to help you develop your applications software. These programs are on a CD that comes with your card and consist of a Setup program and sample programs. The sample programs are in forms suitable for use with BASIC, QuickBASIC, C, and Pascal.
The programs as follows:

FINDBASE: Program locates active and available port addresses.

SETUP: IOD-120 Board Setup Program

BSAMPLES:	*SAMPLE1*	A Quickbasic program that writes a sequence of values to Port A and reads and displays the values in Ports A & B.
	SAMPLE2	A Quickbasic program that displays the bits in Ports A & B and, when an interrupt occurs, polls those same bits.
CSAMPLES:	*SAMPLE1*	A “C” program that writes a sequence of values to Port A and reads and displays the values in Ports A & B.
	SAMPLE2	A "C" program that displays the bits in Ports A & B and, when an interrupt occurs, polls those same bits.
PSAMPLES:	*SAMPLE1*	A “Pascal” program that writes a sequence of values to Port A and reads and displays the values in Ports A & B.

SETUP

This program is supplied with the IOD-72 card as a tool for you to use in configuring jumpers and switches on the card. It is menu-driven and provides pictures of the card on the computer monitor. You make simple keystrokes to select the functions. In turn, the pictures then change to show how the jumpers or switches should be placed to effect your choices.

The setup program is a stand-alone program that can be run at any time. It does not require the IOD-72 to be plugged into the computer for any part of the setup. The program is self-explanatory with operation instructions and on-line help.

PROGRAMMING

The IOD-72 is an I/O mapped device that is easily configured from any language and any language can easily perform digital I/O through the card's ports. This is especially true if the form of the data is byte or word wide. All references to the I/O ports would be in absolute port addressing. However, a table could be used to convert the byte and word data ports to a logical reference. If you are working with VisualBASIC for Windows, then the VBACCESS utility provided with your card provides InPort and OutPort capability.

DEVELOPING YOUR APPLICATION SOFTWARE

If you wish to gain a better understanding of the programs listed in the previous section, then the information in the following paragraphs will be of interest to you. Refer to the data sheets and 8255-5 specification in Appendix A.

A total of 15 address locations are used by the IOD-72; five for each PPI. The PPI's are addressed consecutively with Address bits A3 through A0 (See Address Selection, section 4.) as follows:

ADDRESS SELECTION TABLE

Address	Port Assignment	Operation
Base Address	PA Port 0	Read/Write
Base Address +1	PB Port 0	Read/Write
Base Address +2	PC Port 0	Read/Write
Base Address +3	Control Group 0	Write Only
Base Address +4	PA Port 1	Read/Write
Base Address +5	PB Port 1	Read/Write
Base Address +6	PC Port 1	Read/Write
Base Address +7	Control Group 1	Write Only
Base Address +8	PA Port 2	Read/Write
Base Address +9	PB Port 2	Read/Write
Base Address +A	PB Port 2	Read/Write
Base Address +B	Control Group 2	Write Only
Base Address +C	Enable/Disable Buffer, Grp 0	Write Only
Base Address +D	Enable/Disable Buffer, Grp 1	Write Only
Base Address +E	Enable/Disable Buffer, Grp 2	Write Only

The IOD-72 card uses three 8255-5 PPI's to provide a total of 72 bits input/output capability. The card is designed to use each of these PPI's in Mode 0 wherein:

a. There are two 8-bit ports (A and B) and two 4-bit ports (C Hi and C Lo).
b. Any port can be configured as an input or an output.
c. Outputs are latched.
d. Inputs are not latched.

Each PPI contains a control register. This Write-only, 8-bit register is used to set the mode and direction of the ports. At Power-Up or Reset, all I/O lines are set as inputs. Each PPI should be configured during initialization by writing to the control registers even if the ports

are only going to be used as inputs. Output buffers are automatically set by hardware according to the control register states. Note that control registers are located at base address +3, base address +7, and base address +B. Bit assignments in each of these control registers are as follows:

PPI CONTROL REGISTER BIT ASSIGNMENTS

Bit	Assignment	Function
D0	Port C Lo (C0-C3)	1 = Input, 0 = Output
D1	Port B	1 = Input, 0 = Output
D2	Mode Selection	1 = Mode 1, 0 = Mode 0
D3	Port C Hi (C4-C7)	1 = Input, 0 = Output
D4	Port A	1 = Input, 0 = Output
D5,D6	Mode Selection	01 = Mode 1, 00 = Mode 0
D7	Mode Set Flag	1 = Active

.

Note: Mode 1 cannot be used by the IOD-72 without modification (Consult factory.). Thus, bits D2, D5, and D6 should always be set to "0" and bit D7 to "1".

Similarly, the Group 1 ports and the Group 2 ports can be enabled via the Control Register at Base Address +7 and Base Address +B respectively. The following program fragment in C illustrates the foregoing:

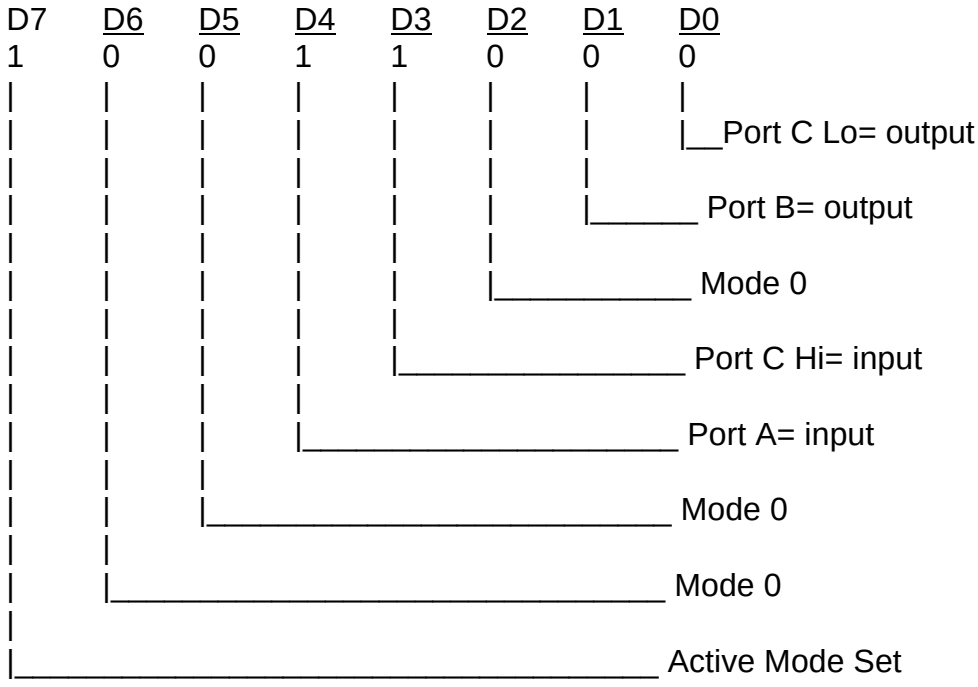
```
const BASE_ADDRESS 0x300

outportb(BASE_ADDRESS+3,0x89);   /*This instruction sets the PPI to Mode 0,
                                 ports A and B as output, and port C as input.
                                 See item b. on the previous page.*/

outportb(BASE_ADDRESS,0);
outportb(BASE_ADDRESS+1,0);      /*These instructions set the initial state of
                                 ports A and B to all zeroes. Port C is not set
                                 because it is configured as an input. See
                                 item c. on the previous page.*/
```

PROGRAMMING EXAMPLE

The following example in BASIC is provided as a guide to assist you in developing your working software. In this example, the card base address is 2D0 hex and I/O lines of Port 0 are to be setup as follows:

Port A = Input
Port B = Output
Port C Hi = Input
Port C Lo = Output

The first step is to configure the control register. Configure bits of the control register as:



This corresponds to 98 hex. If the card base address is 2D0 hex, use the C command to write to the control register as follows:

```
Base=0x2D0;
outportb(Base+3,0x98);
```

To read the inputs at Port A and the upper nybble of Port C, use the C INPUT command:

```
x=inportb(Base);          //Read Port A       x= inportb(Baseaddr);
y=inportb(Base+2)/16;     //Read Port C Hi    y= inportb(Baseaddr+2)>>4;
```

To set outputs high ("1") at Port B and the lower nybble of Port C:

```
outportb(Base+1,0xFF);    //Turn on all Port B bits
outportbBase+2,0xF);      //Turn on all bits of Port C Lo
```

SAMPLE PROGRAMS

The following sample programs are in TURBO-C and TURBO-PASCAL languages. They cover a security system that allows you to monitor the status of 16 switches and to automatically trigger four alarms that can be used to turn on lights, activate a siren, or send a signal to a silent alarm. The alarm system in this demonstration has four arming stations which toggle the alarm on or off. These programs are also provided on the CD that ACCES supplied with your IOD card.

TURBO PASCAL PROGRAM

```
CONST BASEADDR    =$300;  {declare base address for IOD card}
CONST ON          = 1;    {declare some useful constants}
CONST OFF         = 0;    {   "      "    "    "   }

TYPE sensor_array = array[0..15] of integer;      {creates a type of variable used
                                                  {for sensor data}
VAR sensors_at_arm : sensor_array;                {bit-by-bit status of sensors when alarm}
                                                  {is activated. Used to notify user of}
                                                  {open windows, etc}
VAR sensors_now : sensor_now;     {bit-by-bit status of sensors at current time. When}
                                  {compared against sensors_at_arm, indicates}
                                  { break-in if there is a change.}
VAR arming_stations : integer;    {variables representing all four arming stations. If}
VAR old_arming_stations: integer;  {value changes toggle alarm on/off}
VAR hour, min, sec, hun : word;  {variables used to retrieve time}
VAR key : char;                  {useful temporary variable}
VAR i : integer;                 {useful temporary variable, used in loops}
VAR j : integer;                 {  "          "          "         "      "     "  }

procedure initialize_board;      {this procedure sets MODE 0 as active and sets} {Port A, B, and C LO
                                 as input and Port C HI as} {output}
begin
  port[BASEADDR+3] :=$93;        {port[X] is Pascal's method of accessing the port memory. This code}
                                 {sets the port memory at address 303 hex, and the control register, to}
                                 {93 hex because the bit pattern to set the desired mode and port}
                                 {designations is 10010011 which equals 93 hex end; procedure}
                                 {initialize_board}

procedure read_sensors(VAR ary:sensor_ary);
      VAR tempA : byte;          {this procedure fetches data from Ports VAR tempB : byte; A and B}
                                 {and returns a binary representation of each sensor}
begin
      tempA := port[BASEADDR];   {this procedure loads tempA and tempB with corresponding inputs}
                                 {from the IOD card}
      tempB := port[BASEADDR+1];
      for i := 0 to 7 do begin
        if ((tempA shr i) AND ON) > 0 then    {this tests to see if bit #i is ON and sets the}
          ary[i]:=ON                          {corresponding array element to ON}
        else
          ary[i]:=OFF                         {if it is...else, the array element is set to OFF}
      end;
      for i:=0 to 7 do begin
```

```
        if ((tempB shr i) AND ON) > 0 then
          ary[i+8]:=ON                    {in order to get Port B into array elements 8 thru 15,}
        else                              {we add 8 to the bit numbers in the assignment}
          ary[i+8]:=OFF;
        end;
end;{procedure read_sensors}

function get_status:integer;
      var temp:integer;
begin                                {this sets status to the lower nybble of Port}
      temp:=port[BASEADDR+2];        {C;the half defined by Initialize to be input for}
      get_status:=temp AND $0F;      {four arming switches}
end; {function get_arming_status}

procedure ALARM
      var temp:longint;
begin
      sound(2000);                   {this starts the computer's  speaker as siren for the alarm}
      temp:=0
      port[BASEADDR+2]:=$F;          {this sets Port C's lower nybble bits ON }

      repeat
        arming_stations:=get_status                  {this activates four alarm outputs and then}
        if arming_stations <> old_arming_stations then    {toggles Port C Hi's LSB which}
         temp:=2000000000; {disarmed}                     {might be used with an external siren}
         port[BASEADDR+2]:=port[BASEADDR+2] XOR $10;
         temp:=temp+1;
     until temp>=2000000000;
      nosound;
end; {procedure ALARM}

begin
  initialize_board;
  clrscr;
  gotoxy(5,5);
  writeln('This is the IOD card demonstration program. This ');
  writeln('program will simulate an alarm system program for ');
  writeln('sixteen sensors and four arming stations, along with');
  writeln('four separate alarm outputs which could be routed to');
  writeln('a siren, lights, silent alarm,etc');
  writeln;
  writeln('THIS PROGRAM IS INTENDED FOR DEMONSTRATION PURPOSES,');
  writeln('ONLY AND IS NOT MEANT TO BE USED AS AN ACTUAL ALARM ');
  writeln('SYSTEM.');
  writeln;writeln;
  writeln('Press any key to begin program.');
  key:readkey;
  old_arming_stations:=get_status;  {this loads the status of the arming switches}
  repeat                            {at the time the program is first activated. A}
    clrscr;                         {change in status would indicate arming}

    read_sensors(sensors_now);      {this reads the current status of the sensors,}
    for i=0 to 15 do begin          {which is then displayed to indicate open windows, etc.}
      if sensors_now[i]=OFF
        writeln('Sensor #', i, 'is open');
    end;
    writeln;
    writeln('Press ESC to re-scan, RETURN to begin alarm scanning.');
```

```
  key:=readkey;
 until key=#13;          {the repeat/until loop gives the user  an opportunity}
                         {to shut open windows or doors, and then re-scan the sensors}
 clrscr;

WHILE TRUE do begin              {this WHILE is used to form an infinite loop}

 Writeln('Waiting to be armed, or press any key to halt program.');
 repeat                          {this repeat/until-loop continues until arming}
  arming_stations:=get_status;   {station status  changes, indicating arming or}
  if key pressed then halt(1);   {until a key is pressed terminating the program.}
 until arming stations <> old arming stations;
 sound(900);                     {short tone indicating that alarm has been armed}
 delay(300);                     {  "      "        "        "     "     "     "      " }
 nosound;                        {  "      "        "        "     "     "     "      " }
 writeln('Alarm system will activate in 15 seconds');
 read_sensors(sensors_at_arm);
 old_arming_stations : get_status;
 gettime(hour,min,sec,hun);          {this code reads the system clock for the current}
 i:=sec+15;                      {time which is used to delay for 15 seconds}
  if i > 60 then i :=i-60;
 repeat
  gettime(hour,min,sec,hun);
 until sec = i;                  {end of delay loop}
 writeln;
 writeln('ALARM SYSTEM ACTIVE AND ARMED');
 sound(900);                     {short tone indicating that alarm is fully activated}
 delay(300);
 no sound;
 j:=0                        {the following code compares current status of sensors against status when}
                             {armed to determine if break-in has occurred..any change indicates break-in}
 repeat
  read_sensors(sensors_now);
  for i:= 1 to 16 do begin
    if sensors_now[i-1] <> sensors_at_arm[i-1]then
      j:=1;
  end;{for}
  arming_stations: get_status;
  if arming_stations <> old_arming_stations then
    j:= -i;                  {flag used to signal that alarm is de-activated}
 until j <> 0;
 if j = -1 then begin        {if j was set to -1 in the above loop, then alarm is}
          {de-activated}
  gettime(hour,min,sec,hun);
  writeln('Alarm deactivated at ', hour,':',min,':',sec);

  sound(900);                {the following code chirps the speaker to indicate disarming}
  delay(100); nosound;
  delay(50); sound(900);
  delay(100); nosound;
  nosound;
 end                         {end of disarming routine}
 else {if alarm}begin
  writeln('Sensor #', j,' has been activated!!');
  gettime(hour,min,sec,hun);
  writeln('The time of alarm is ',hour,':',min,':',sec);
  ALARM;
 end;                        {else}
```

```
  end;                    {WHILE   this "end" sends the program back }
                          {to wait to be re-armed}
end.
```

TURBO-C PROGRAM

```
#define BASEADDR 0x300  /*declare base address for IOD card*/
#define ON       1      /*create useful constant*/
#define OFF      0      /*    "       "          "     */
#include "stdio.h"
#include "conio.h"
#include "time.h"
#include "dos.h"

int sensors_at_arm[15];
int sensors_now[15];        /*bit-by-bit status of sensors at current time*/
                            /*When compared against status of sensors*/
                            /* at arm, indicates break-in if there is a change.*/

int arming_stations;        /*variables representing all four arming stations*/
int_old_arming_stations;        /*If the value changes, toggle alarm ON/OFF.*/
char key;                   /*useful temporary variable*/
int i;                      /*useful temporary variable used in loops*/
int j;                      /*useful temporary variable*/

initialize(){
  outportb(BASEADDR+3,0x93);      /*outportb(addr,byte) is C's method of accessing*/
                                  /*port memory. This procedure sets Port A, B, and*/
                                  /*C LO as inputs and Port C HI as outputs.*/
                                  /*Address* 303 hex is the control register. The bit*/
                                  /*pattern needed to  set the desired mode and port*/
                                  /*designation is 10010011 = 93 hex*/
} /*procedure initialize*/

read_sensors(int *ary){
unsigned char tempA;
unsigned char tempB;
        tempA = inportb(BASEADDR);
        tempB = inportb(BASEADDR+1);
        for(i-0;i<8;i++){
          if((tempA>> i) & ON){     /*this determines if bit #i is on and sets the corre-*/
            *ary++=ON;}             /*sponding array element to ON if it is. If not, sets*/
          else{                 /*sets the array element to OFF */
            *ary++=OFF;}

        }
        for(i=0;i<8;i++){
          if((tempB>> i) & ON){
            *ary++=ON; }
          else
            *ary++=OFF; }
        }
} /*procedure read_sensors*/

get_status(){
        int temp;
```

```
    temp=inportb(BASEADDR+2);       /*this sets status to the lower half of Port C, the*/
                                    /*half defined in Initialize to be input, for four*/
                                    /*arming switches.*/
    return temp & 0x0F;
  } /*function get_arming_status*/
  ALARM(){
  long int temp=0;
    sound(2000);                    /*this starts the computer's speaker*/

    outportb(BASEADDR+@,0xF0);      /*this sets Port C upper nybble bits*/
                                    /*to ON (1111 binary = F hex).*/
    do{
      arming_stations=get_status();             /*this activates four alarm outputs and */
      if(arming_stations !=old_arming_stations)  /*then toggles Port C Hi LSB which*/
        temp=2000000000; /*dis-armed*/          /*maybe used with an ext'l speaker*/
      outportb(BASEADDR+2,inportb(BASEADDR+2)^0x10);
    }while(temp++ !=2000000000);
    nosound();
  } /*procedure ALARM*/


main()
    {

    time_t start;
      initialize();
      clscr();
      goto(5,5);
      printf("This IOD-card demonstration program simulates an alarm\n");
      printf("system program for 16 sensors, four arming stations and\n");
      printf("four separate alarm outputs which could be routed to a\n");
      printf("siren, lights, silent alarm, etc.\n");
      printf("\n");
      printf("THIS PROGRAM IS FOR DEMONSTRATION PURPOSES ONLY, AND\n"
      printf("IS NOT MEANT TO BE USED AS AN ACTUAL ALARM SYSTEM.\n");
      printf("\n");printf("n");
      printf("Press any key to begin program.\n");
      key=getch();
      old_arming_stations=get_status();
      do{
        clrscr();
        read_sensors(sensors_now);
        for(i=0;i<=15;i++){
          if (!sensors_now[i]) printf("Sensor #%d %s\n,i,"is open");
        }
        printf("\n");
        printf("Press ESC to re-scan, RETURN to begin alarm scanning.");
        key=getch();
      }while(key!=13);
      clrscr();
     for(;;){                           /*this creates an infinite loop*/
      printf("Waiting to be armed. Press any key to halt program.\n");
      do{
        arming_stations=get_status();
        if(kbhit()) abort(0);
      }while(arming_stations== old_arming_stations);

sound(1000); delay(300);
```

```
nosound();
printf("Alarm system will activate in 15 seconds");
read_sensors(sensors_at_arm);
old_arming_stations=get_status();
start=time(NULL);
do{
}while(difftime(time(NULL),start) !=15);
printf("\n");
printf("ALARM SYSTEM ACTIVE AND ARMED\n\n");
sound(900); delay(300);
nosound();
j=0;
do{
    read_sensors(sensors_now);
    for(i=1;i<=16;i++){
      if(sensors_now[i-1] !=sensors_at_arm[i-1])
          j=i;
  } /*for*/
  arming_stations = get_status();
  if (arming_stations != old_arming_stations)
  j=-1                    /*flag used to signal alarm is de-activated*/
  while(!j);
  if(j == -1){
    start=time(NULL);
    printf("Alarm deactivated at %s,(asctime(gmtime(&start))));
    sound(900); delay(300);
    nosound(); delay(50);
    sound(900); delay(100);
    nosound();
  }  else {
    printf("Sensor #%d has been activated!!\n\n",j);
    start=time(NULL);
    printf("The time of alarm is %s", asctime(gmtime( &start)));
    old_arming_stations=get_status();
    ALARM();
  } /*else*/
 } /* for(;;) this "end" used to send program back to await re-arm*/
}
```

SHARING INTERRUPTS ON THE ISA BUS

As noted on page 3-1, IOD-72 can facilitate an external interrupt via bit C3 at each 24-bit group. On occasion, however, a system application will require more interrupt levels than are available on the ISA bus. While not recommended, IRQ sharing is possible. Each card that is going to share an IRQ ***must strictly adhere to a special standard for accessing the IRQ line as follows:***

1. The interrupt must be held in a high impedance state until asserting an interrupt.

2. The interrupt must be asserted in the form of a low signal lasting at least 500 nanoseconds followed by a rising edge and then immediately returning to a high impedance condition.

3. The card must contain a status register or flag of some kind to indicate that it generated the interrupt. There is an exception to this rule. This is the case where only one card of those sharing the interrupt level does not provide a status bit to indicate that it asserted the interrupt but is otherwise capable of sharing the IRQ. In this case, it may share the interrupt level with other cards if (a) it is the only card on that IRQ level that does not have a status bit and (b) it is installed onto the IRQ vector first. (This makes it the last card to be called in the vector chain.) This scheme will work because it can be assumed that if every other card in the vector chain did not cause the interrupt, then the last card must be the one that did.

Note that, if two cards assert the IRQ line within 500 nanoseconds of each other, the second card in the ISR chain will not be serviced. It's possible to alleviate this problem by writing a single ISR that can detect the bit flag on every card and therefore detect the fact that two (or more) cards report generating an interrupt even though only one interrupt was processed by the CPU.

CONNECTOR PIN ASSIGNMENTS

Three 50-pin headers are provided on the IOD-72; one for each group of 24 I/O lines. The mating connector is an AMP type 1-746285-0 or equivalent. Connector pin assignments are listed below. Notice that every second line is grounded to minimize crosstalk between signals.

Assignment		Pin		Assignment	Pin
Port C Hi	PC7*	1		Ground	2
Port C Hi	PC6	3		"	4
Port C Hi	PC5	5		"	6
Port C Hi	PC4	7		"	8
Port C Lo	PC3**	9		Ground	10
Port C Lo	PC2	11		"	12
Port C Lo	PC1	13		"	14
Port C Lo	PC0	15		"	16
Port B	PB7	17		Ground	18
Port B	PB6	19		"	20
Port B	PB5	21		"	22
Port B	PB4	23		"	24
Port B	PB3	25		"	26
Port B	PB2	27		"	28
Port B	PB1	29		"	30
Port B	PB0	31		"	32
Port A	PA7	33		Ground	34
Port A	PA6	35		"	36
Port A	PA5	37		"	38
Port A	PA4	39		"	40
Port A	PA3	41		"	42
Port A	PA2	43		"	44
Port A	PA1	45		"	46
Port A	PA0	47		"	48
+5 VDC		49		Ground	50

NOTES:
* This line is an I/O port and also an Interrupt Enable
** This line is an I/O port and also an User Interrupt

SPECIFICATION

Features

72 Channels of Digital Input/Output.
All 72 I/O Lines Buffered on the Board.
Four and Eight Bit Groups Independently Selectable for I/O.
Hysteresis correction and 10KΩ Pull-Ups on I/O Lines.
Interrupt and Interrupt-Disable Capability.
+5V Supply Available to the User.
Compatible with Industry-Standard I/O Racks like Opto-22, Potter & Brumfield, etc.

Digital Inputs

Logic High: 2.0 to 5.0 VDC.
Logic Low: -0.5 to +0.8 VDC.
Input Load (Hi): 20 uA.
Input Load (Lo): -200 uA.

Digital Outputs

Logic High: 2.5 VDC min., source 15 mA.
Logic Low: 0.5 VDC max., sink 24 mA.
(64 mA optional)

Power Output: +5 VDC from computer bus (ext. 1A fast-blow fuse recommended).

Power Required: +5 VDC at 350 mA typical.

Size: 7.5" Long.

Environmental:

Operating Temperature: 0 degr. to 60 degr. C.
Storage Temperature: -50 degr. to +120 degr. C.
Humidity: 0 to 90% RH, non-condensing.

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.

APPENDIX A

PROGRAMMABLE PERIPHERAL INTERFACE DATA SHEETS

The data sheets in this Appendix are provided to help your understanding of the 8255-5 PPI which is made by a number of companies. These sheets are reprinted with permission of Mitsubishi Electric Corp. (Copyright 1987).

The information, diagrams, and all other data included are believed to be correct and reliable. However, no responsibility is assumed by Mitsubishi Electric Corporation for their use, nor for any infringements of patents or other rights belonging to third parties which may result from their use. Values shown on these data sheets are subject to change for product improvement.