

1781-PXB721
Discrete I/O Interface
User's Manual

Western Reserve Controls, Inc.

Although every effort has been made to insure the accuracy of this document, all information is subject to change without notice. WRC takes no liability for any errors in this document or for direct, indirect, incidental or consequential damage resulting from the use of this manual.

Document PUB 28.0
Rev 1.50
November 2000

Copyright © 1996-2000 WRC

Western Reserve Controls, Inc.

1485 Exeter Road
Akron OH 44306
330-733-6662 (Phone)
330-733-6663 (FAX)
sales@wrcakron.com (Email)
<http://www.wrcakron.com> (Web)

SmartMux, PXB721 and WRC are trademarks of Western Reserve Controls, Inc.
IBM PC, PC/XT, and PC/AT are registered trademarks of the International Business Machines Corporation.
All other trademarks are property of their respective companies.

TABLE OF CONTENTS

1. INSTALLATION 1
2. FUNCTIONAL DESCRIPTION..... 3
3. OPTION SELECTION..... 5
4. ADDRESS SELECTION..... 7
5. SOFTWARE..... 9
6. PROGRAMMING..... 12
7. CONNECTOR PIN ASSIGNMENTS 22
8. SPECIFICATIONS 23
APPENDIX A PPI DATA 24

TABLE OF TABLES

Table 1 Standard Address Assignments for PC and PC/XT Computers 7
Table 2 Address Set-up Switches..... 8
Table 3 Switch Selection Example 8
Table 4 Address Selection Table 12
Table 5 I/O Control Register Definitions 13
Table 6 Connector Pin Assignments..... 22

TABLE OF FIGURES

Figure 1 1781-PXB721 OPTION SELECTION MAP..... 6

1. INSTALLATION

BACKING UP THE DISK

The software provided with the PXB721 is on a 3.5" floppy. As with any software package, you should make backup copies for everyday use and place your original master diskette 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:
```

In a two-disk system the command is:

```
COPY A:*.* B:
```

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

HARD DISK USERS

The files contained on the master disk may also be copied onto your hard disk. Files contained on the disk are stored in separate directories. These are:

FINDBASE:	Contains tools to find a working card address.
PSAMPLES:	Contains Pascal samples and the Pascal-linkable driver.
CSAMPLES:	Contains "C" samples and the C-linkable driver.
BSAMPLES:	Contains the BASIC and QuickBASIC samples as well as the binary Q and linkable drivers.
VB_WRC:	Contains VisualBASIC sample and VisualBasic-linkable driver.

To install on a hard disk:

- 1) Place the master diskette into a floppy drive.
- 2) Change the active drive to the drive that contains the master diskette. This is usually drive A.
- 3) Type INSTALL and follow the screen prompt.

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 programs

called PXB721ST and FINDBASE provided on diskette with the card. They supply 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.

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 3.
5. Select the base address on the card. See ADDRESS SELECTION, section 4
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.

2. 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 and Pull-Ups on I/O Lines.
- Interrupt and Interrupt-Disable Capability.
- +5V Supply Available to the User.
- Compatible with WRC Industry Standard I/O Racks and other, such as 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 Closure or TTL, DTL, CMOS Logic
- Driving Indicator Lights or Recorders

The 1781-PXB721 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 24 mA (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 the 1781-PXB721 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 external circuits.

The PXB721 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

1781-PXB721 USER'S MANUAL

Reference Manual. For additional information about setting the base address of 1781-PXB721, see the Address Selection Section of this manual.

Utility software provided on diskette with the 1781-PXB721 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 Software Section of this manual. In addition a sample program and utility driver are provided for use with VisualBASIC for windows.

1781-PXB721 BLOCK DIAGRAM

Typical of three sections

3. OPTION SELECTION

Refer to the setup programs on the diskette provided with the card. Also, refer to the 1781-PXB721 card BLOCK DIAGRAM on the previous page and the OPTION SELECTION MAP on the following pages when reading this Section of the manual.

Board Address selection is covered both by the setup program and by the Address Selection Section 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 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 1781-PXB721. 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.

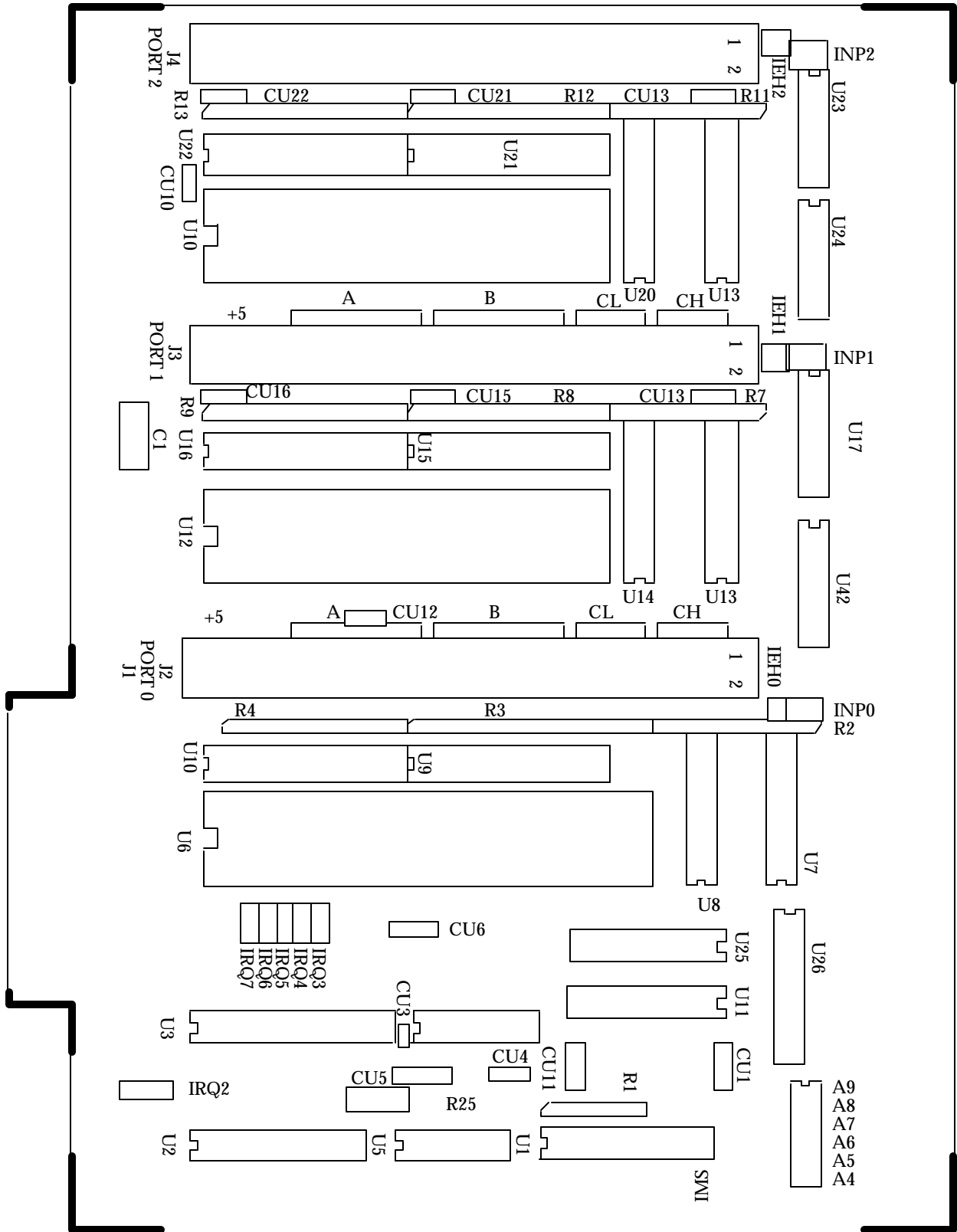


Figure 1 1781-PXB721 OPTION SELECTION MAP

4. ADDRESS SELECTION

The 1781-PXB721 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 options cannot share the same address. If in doubt where to assign the base address of the PXB721, refer to the tables below and the FINDBASE program to find an available address in your system.

Table 1 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 Slot
220-24F	Reserved
278-27F	Reserved
2F0-2F7	Reserved
2F8-2FF	Asynchronous Communication (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
3F0-3F7	Diskette
3F8-3FF	Asynchronous Communication (primary)

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

To set desired board addresses, refer to the illustrated Board Address setup program on the Utility diskette provided with the card. Type the desired address in hexadecimal code and 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-type logic. (Assign '0' to all ADDRESS SETUP switches turned ON, and assign '1' to all ADDRESS SETUP switches turned OFF.)

Table 2 Address Set-up Switches

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 3, PROGRAMMING.

Table 3 Switch Selection Example

Hex representation	2+0=2		8+4+0+1=13=D(hex)			
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. (label)	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.

5. SOFTWARE

WESTERN RESERVE CONTROLS supplies several programs to support the 1781-PXB721 digital I/O card and, also, to help you develop your applications software. These programs are on a diskette that comes with your card and consist of a Setup program and three sample programs. The sample programs are in forms suitable for use in BASIC, QuickBASIC, C, and Pascal. The programs are as follows:

- * PXB721ST 1781-PXB721 Board Setup Program.
NOTE: This program should run directly out of DOS
- * FINDBASE. . . . Reports active and available address locations in your computer for assignment as the 1781-PXB721 base address.
- * SAMPLE1. . . . A program that writes a sequence of values to port A and reads and displays the values in Ports A & B.
- * SAMPLE2. . . . A program that displays the bits in Ports A & B and, when an interrupt occurs, polls those same bits.
- * WRC_VB. . . . A driver to enable use of the 1781-PXB in VisualBASIC for Windows. Also includes sample.

PXB721ST

This program is supplied with the 1781-PXB721 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. Making simple keystrokes to select functions. In turn, the pictures change to show jumpers and switch settings. The setup program is a stand-alone program that must be run from DOS. It does not require the 1781-PXB721 to be plugged into the computer for any part of the setup. The program is self-explanatory with operation instructions and on-line help.

To run this program, at the DOS prompt, enter PXB721ST followed by the [ENTER].

PXB721 DEMONSTRATION

The demonstration is written using the Visual Basic 3.0 driver. It is able to access all of the board functions in Tri-State and buffer enabled mode. See the on-line help and follow the standard programming directions.

VISUALBASIC UTILITY DRIVER

WESTERN RESERVE CONTROLS now provides extensions to the VisualBASIC language on the diskette provided with your card. The extensions are in a directory named VB_WRC. These extensions are in the form of a .DLL, a .GBL, and a VisualBASIC sample. Together these files allow you to access the port and main memory space in a fashion similar to other standard programming languages.

To use these files in a VisualBASIC program, you must create a .MAK file similar to the sample provided, or modify your existing project file. The VB_WRC.GBL file must be included (File | Add File). The Sample looks for VB_WRC.DLL in the Windows directory.

The additional VisualBASIC functions are:

InPortb

Function: Reads a byte from a hardware port. Due to limitations of VisualBASIC, the number is returned as an integer.

Declaration: function InPortb(*byval address* as integer) as integer

InPort

Function: Reads an integer from a hardware port. This function returns the 16-bit value obtained from reading the low byte from *address* and the high byte from *address+1*.

Declaration: function InPort(*byval address* as integer) as integer

OutPortb

Function: Writes the lower eight bit of *value* to the hardware port at *address*. This function returns the value output.

Declaration: function OutPortb(*byval address* as integer, *byval value* as integer) as integer.

OutPort

Function: Writes all 16 bits of *value* to the hardware port at *address*. This function returns the output value.

Declaration: function OutPort(*byval address* as integer, *byval value* as integer) as integer

Peek

Function: Reads a byte from main memory (DRAM)

Declaration: function Peek(byval segment as integer, byval offset as integer) as integer.

Poke

Function: Writes the lower eight bits of *value* to *segment:offset*.

Declaration: function Poke(byval segment as integer, byval offset as integer, byval value as integer) as integer.

Note that in all the above functions, an inherent limitation of VisualBASIC makes the values sent less intuitive. All integers are signed numbers, wherein data are stored in two's complement form. An alternative is to perform all assignments in hexadecimal, rather than decimal form.

Before a program will execute, the .GBL file must be modified to include the path to VB_WRC.DLL as appropriate for your system. Merely replace the statement "VB_WRC.DLL" with "*drive:path* \VB_WRC.DLL".

As an alternative to changing the source code, VB_WRC.DLL can be copied into the Windows directory. This will allow multiple programs to find the same .DLL without having to know where it is located. Just leave off all references to a path in the .GBL file as shown in the sample.

6. PROGRAMMING

The 1781-PXB721 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 VB_WRC utility provided on the diskette with your card provides InPort and OutPort capabilities.

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 PXB721; five for each PPI. The PPI's are addressed consecutively with address bits A3 through A0 (See Address Selection) as follows:

Table 4 Address Selection Table

Address	Port Assignment	Operation
Base Address +0	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 port 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 port 1	Write Only
Base Address +8	PA port 2	Read/Write
Base Address +9	PB Port 2	Read/Write
Base Address +A	PC port 2	Read/Write
Base Address +B	Control port 2	Write Only

The 1781-PXB721 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 register 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:

Table 5 I/O Control Register Definitions

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, 1X-Mode2
D7 - Mode Set Flag:	1-Active	

Note: Mode 1 cannot be used by the 1781-PXB721 without modification (Consult factory.). Thus, bits D2, D5, and D6 should always be set to "0" and bit D7 to "1".

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:


```

VAR sensors_at_arm : sensor_array
    {creates a type of variable used for}
    {sensor data}

VAR sensors_now : sensor_now;
    {bit-by-bit status of sensors when}
    {alarm is activated. Used to notify}
    {user of open windows, etc.}
    {bit-by-bit status of sensors at}
    {current time. When compared}
    {against sensors_at_arm, indicates}
    {break-in if there is a change.}

"AR arming_stations : integer;
VAR old_arming_stations: integer ;
    {variables representing all four}
    {arming stations. If value changes}
    {toggle alarm on/off}

VAR hour, min, sec, hun : word;
VAR key : char;
VAR i : integer;
VAR j : integer;
    {variables used to retrieve time}
    {useful temporary variable}
    {useful temporary variable, used in loops}
    {" " " " " " " "}

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, the control}
    {register, and to 93 hex because the bit}
    {pattern to set the desired mode and port}
    {designations is 1001001 which equals}
    {93 hex}

    end; {procedure initialize_board}

procedure read_sensors(VAR ary:sensor_ary);
    VAR tempA : byte;
    VAR tempB :byte;
    {This procedure fetches data from ports}
    {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 PXB card}

    tempB := port[BASEADDR+1];
    for i :=0 to 7 do begin
        for i :0 to 7 do begin
            if ((tempA shr i) AND ON) > 0 then    {This tests to see
                ary[i]:=ON                        {if bit #i is on and sets}
            else                                    {the corresponding array}
                ary[i]:=OFF                        {element to ON. If it is.}

        end;
    end;
    {...else, the array element is}
    {set to OFF}

```

```

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}
  else                                           {array, elements 8 thru 15,}
    ary[i+8]:=OFF;                               {we add 8 to the bit numbers}
  end;                                           {in the assignment}
end; {procedure read_sensors}

function get_status:integer;
  var temp:integer;
begin
  temp:=port[BASEADDR+2];                       {This sets status to the lower}
  get_status:=temp AND $0F;                     {nybble of Port C; the half}
  end; {function get_arming_status}             {defined by Initialize to be}
                                              {input, for 4 arming switches}

procedure ALARM
  var temp:longint;
begin
  sound (2000);                                 {This starts the computer's}
                                              {speaker which acts as siren}
                                              {for the alarm}

  temp:=0
  port[BASEADDR+2]:=$F;                         {This sets Port C's lower}
                                              {nybble bits to ON}

  repeat
    arming_stations:=get_status                 {This activates}
  if arming_stations <> old_arming_stations then {4 alarm outputs and then}
    temp:=2000000000; {disarmed}               {toggles Port C Hi's LSB}
    port[BASEADDR+2]:=port[BASEADDR+2] XOR $10; {which might be used with}
    temp:=temp+1;                               {used with external}
  until temp>=2000000000;                       {siren}
  nosound;
end; {procedure alarm}
begin
  initialize_board;
  clrscr;
  gotoxy(5,5);
  writeln('This is the PXB 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, siren 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. ');

```



```

read_sensors(sensors_now);
for i:=1 to 16 do begin
  if sensors_now[i-1]then
    j:=1;
end;;{for}
arming_stations: get_status;
if arming_station <> old_arming_stations then
  j:= -1;                                {flag used to signal that alarm is}
                                          {deactivated}

until j <> 0;
if j = -1 then begin                      {if j was set to -1 in the above loop, then}
                                          {alarm is deactivated}

  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 alarming routine}
else {if alarm}begin
  writeln('sensor #', j, ' has been activated!!');
  gettime(hour,min,sec,hun);
  writeln('The time of alarm is ',hour,':',mon,':',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 PXB 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*/
                                          /*status of alarms at arm, indicates */

```

```

int_armed_stations; /*break-in if there is a change.*/
int_old_armed_stations; /*variables representing all four arm-*/
/*ing 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 = importb(BASEADDR);
    tempB = importb(BASEADDR+1);
    for(i=0;i<B;i++){
        if((tempA>> i) & ON){ /*this determines if bit #1 is on and*/
/*sets the corresponding array element*/
            *ary++=ON;}
        else{ /*to ON if it is. If not, sets the*/
/*array element to OFF*/
            *ary++=OFF;}
    }
    for(i=0;i<B;i++){
        if((tempB>> i) & ON){
            *ary++=ON;}
        else
            *ary++=OFF;}
} /*procedure read_sensors*/

get_status(){
    int temp;
    temp+importb(BASEADDR+2); /*this sets status to the lower half of
/*Port C, the half defined in In- */
/*italize to be input, for four arming*/
/*switches.*/

    return temp & 0x0F;
} /*function get_armed_status*/
ALARM(){
    long int temp=0;

```

```

sound(2000);                                /*this starts the computer's speaker*/

outportb((BASEADDR+@,0xF0); /*this sets Port C under nybble bits*/
                                                /*to ON (1111 binary = F hex).*/

do{
    arming_status=get_status();                /*This activates 4 alarm*/
    if(arming_status !=old_arming_stations)    /*outputs and then toggles*/
        temp=2000000000; /*dis-armed*/        /*Port C high LSB switch*/
    outportb(BASEADDR+2,inportb(BASEADDR+2)^0x10); /*might be*/
}while(temp++ !=2000000000);                  /*used with an electronic*/
    nosound();                                /*speaker*/
} /*procedure ALARM*/

main()
{

time_t start;
    initialize();
    clrscr();
    goto(5,5);
    printf("This PXB-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,");
    printf("AND IS\n NOT MEANT TO BE USED AS AN ACTUAL ALARM");
    printf("SYSTEM.\n");
    printf("\n");printf("\n");
    printf("Press any key to begin program.\n");
    key=getch();
    old_arming_station=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(;;){
printf("Waiting to be armed. Press any keyto 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 deactivated*/

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_station=get_status();
    ALARM();
} /*else*/
} /*for(;;) this "end" used to send program back to await re-arm*/}
```

7. CONNECTOR PIN ASSIGNMENTS

These 50-pin headers are provided on the PXB721; one for each group of 24 I/O lines. The mating connector is an AMP type 1-499776-0 orequivalent. Connector pin assignments are listed below. Notice that every second line is grounded to minimize crosstalk between signals.

Table 6 Connector Pin Assignments

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

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

8. SPECIFICATIONS

Features

- * 72 buffered input/output lines.
- * Hysteresis on input lines improves noise margins
- * Pull-ups on I/O lines for CMOS and contact-closure compatibility.

Digital Inputs

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

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 200 mA typical.

Size: 7.15" 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

APPENDIX A PPI DATA

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.