

ARTISAN®
TECHNOLOGY GROUP

Your **definitive** source for quality pre-owned equipment.

Artisan Technology Group
(217) 352-9330 | sales@artisantg.com | artisantg.com

Full-service, independent repair center
with experienced engineers and technicians on staff.

We buy your excess, underutilized, and idle equipment
along with credit for buybacks and trade-ins.

Custom engineering
so your equipment works exactly as you specify.

- Critical and expedited services
- In stock / Ready-to-ship
- Leasing / Rentals / Demos
- ITAR-certified secure asset solutions

Expert team | Trust guarantee | 100% satisfaction

All trademarks, brand names, and brands appearing herein are the property of their respective owners.

Visit our website - *Click HERE*

OPERATIONS MANUAL
PCM-UIO96B-G

WinSystems reserves the right to make changes in the circuitry and specifications at any time without notice.
©Copyright 2011 by WinSystems. All Rights Reserved.

REVISION HISTORY

P/N 403-0285-000

ECO NUMBER	DATE CODE	REVISION
ORIGINATED	990818	A
99-95	991104	A1
02-76	021203	A2
03-31	030410	A3
07-36	070319	A4
10-82	111208	A5
	110608	A6

TABLE OF CONTENTS

SECTION NUMBER	PARAGRAPH TITLE	PAGE NUMBER
1	General Information	
1.1	Features	1-1
1.2	General Description	1-1
1.3	Specifications	1-2
2	PCM-UIO96B-G Technical Reference	
2.1	Introduction	2-1
2.2	I/O Address Selection	2-1
2.3	Interrupt Routing Selection	2-2
2.4	I/O Connector Pinout	2-3
2.5	PC/104 Bus Interface	2-5
2.6	WS16C48 Register Definitions	2-6
2.7	Connector/Jumper Summary	2-7
3	PCM-UIO96B-G Programming Reference	
3.1	Introduction	3-1
3.2	Function Definitions	3-1
3.3	Sample Programs	3-6
APPENDIX A	PCM-UIO96B-G Parts Replacement Guide	
APPENDIX B	I/O Routine & Sample Program Source Listings	

1 GENERAL INFORMATION

1.1 FEATURES

- 96 Digital I/O Lines
- PC/104 16-bit Interface
- Each line can serve as an input or an output
- Readback capability on all output lines
- Programmable polarity event sense on 48 lines
- Compatible with standard I/O racks
- +5 Volt only operation
- Extended temperature range -40℃ to +85℃

1.2 GENERAL DESCRIPTION

The PCM-UIO96B-G is a highly versatile RoHS-compliant PC/104 module providing 96 lines of digital I/O. It is unique in its ability to monitor 48 lines for both rising and falling digital edge transitions, latch them, and then issue an interrupt to the host processor. The application interrupt service routine can quickly determine, through a series of interrupt identification registers, the exact port(s) and bit(s) which have transitioned. The PCM-UIO96B-G utilizes the WinSystems' WS16C48 ASIC High Density I/O Chip (HDIO). The first 48 lines are capable of fully latched event sensing with the sense polarity being software programmable. Four 50-pin I/O connectors allow for easy mating with industry standard I/O racks.

1.3 SPECIFICATIONS

1.3.1 Electrical

Bus Interface:	PC/104 16-Bit
VCC:	+5V +/-5% @ 24mA typical with no I/O connections
I/O Addressing:	12-bit user jumperable base address. Each board uses 32 consecutive addresses

1.3.2 Mechanical

Dimensions:	3.6” x 3.8”
PC Board:	FR-4 Epoxy glass with 2 signal layers, 2 power planes, screened component legend and plated through holes.
Jumpers:	0.025” square posts on 0.100” centers
Connectors:	50 Pin 0.10” grid RN type IDH-50-LP

1.3.3 Environmental

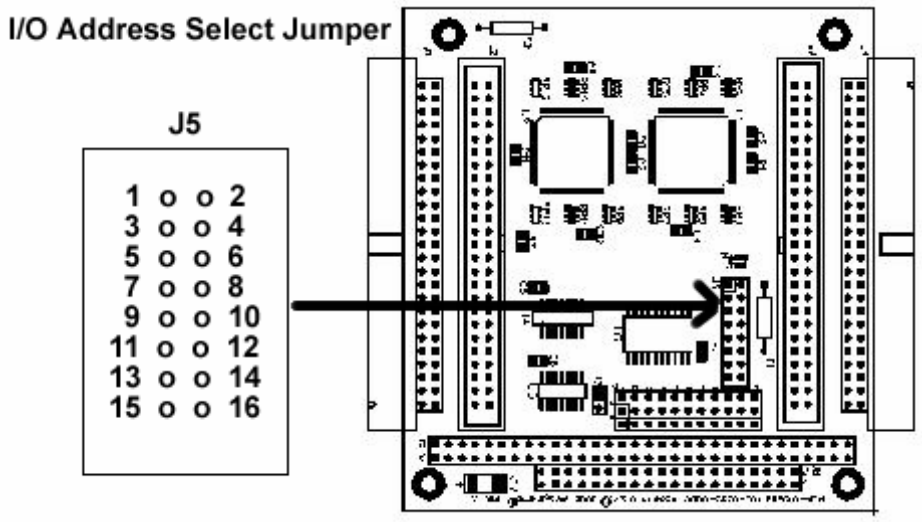
Operating Temperature:	-40℃ to +85℃
Non Condensing Humidity:	5% to 95%

2 PCM-UIO96B-G TECHNICAL REFERENCE

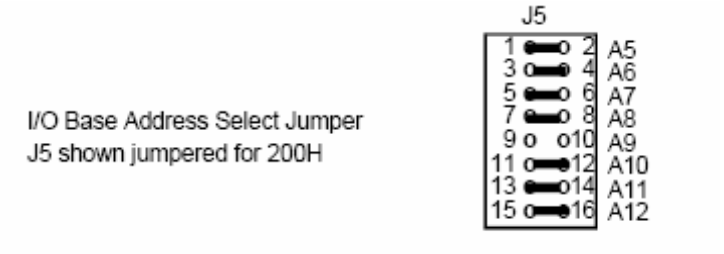
2.1 INTRODUCTION

This section of the manual is intended to provide the necessary information regarding configuration and usage of the PCM-UIO96B-G. WinSystems maintains a Technical Support Group to help answer questions regarding configuration, usage, or programming of the board. For answers to questions not adequately addressed in this manual, contact Technical Support at (817) 274-7553 between 8AM and 5PM Central Time.

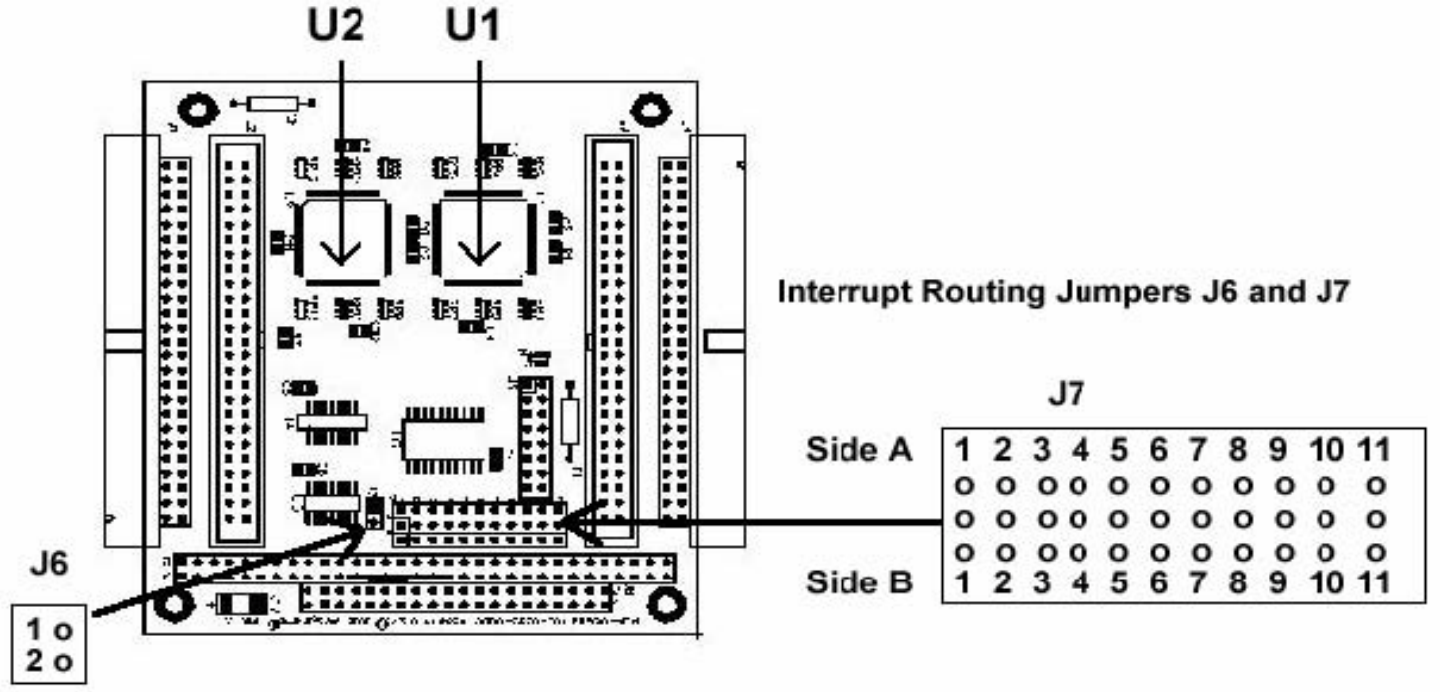
2.2 I/O ADDRESS SELECTION



The PCM-UIO96B-G requires 32 consecutive I/O addresses beginning on a 32 byte boundary. The jumper block at J5 allows for user selection of the base address. Address selection is made by placing a jumper on the jumper pair for the address bit if a '0' is desired or leaving the address bit open if a '1' is required for the desired address. U2 is located at the Base Address, U1 is located at the Base Address plus 10 Hex. The illustration below shows the relationship between the address bit and the jumper positions and a sample jumpering for a base address of 200H.



2.3 INTERRUPT ROUTING SELECTION



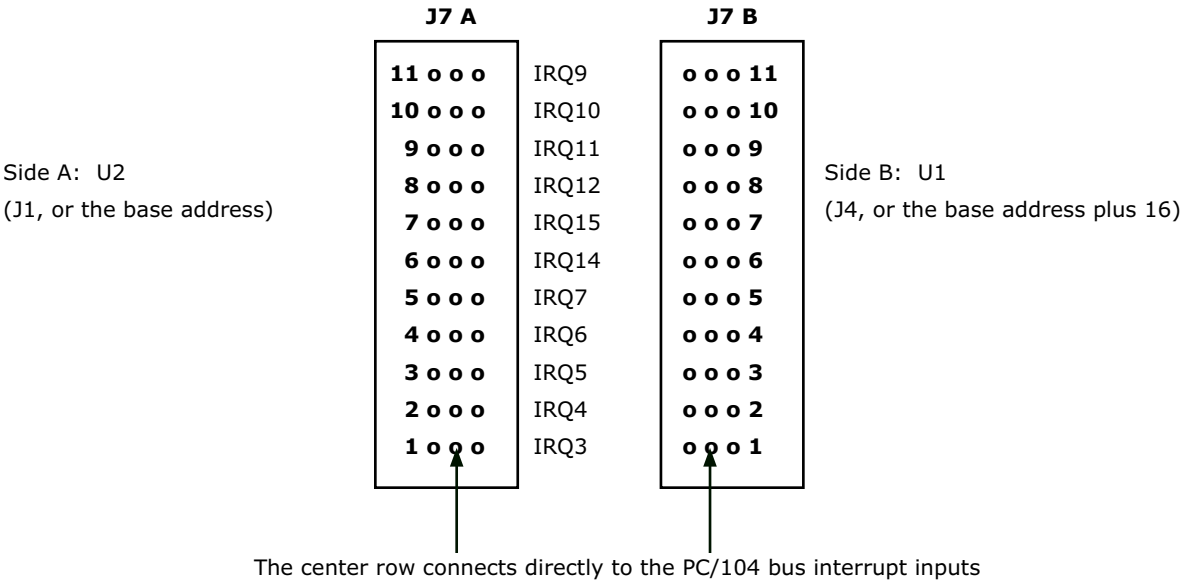
2.3.1 Shared Interrupts

For shared interrupts on the PC/104 bus, J6 must be jumpered pins 1-2. For individual interrupts, J6 should remain unjumpered as shown below.



2.3.2 Interrupt Selection

When desired, the PCM-UIO96B-G can generate an interrupt on up to 48 different lines each with its own polarity select. This interrupt can be routed to the PC/104 bus via the jumper at J7. J7 is a three row eleven pin interrupt routing block, side A directs interrupts to connector J1, side B directs interrupts to connector J4, and the center row connects directly to the PC/104 bus interrupt inputs.



2.4 I/O CONNECTOR PINOUT

The PCM-UIO96B-G routes it 96 lines to four 50-pin IDC connectors at J1, J2, J3 and J4. The pin definitions for J1, J2, J3 and J4 are shown here.

U2 = BASE_ADDR

J1

Port 2 Bit 7	**1 o o 2**	GND
Port 2 Bit 6	**3 o o 4**	GND
Port 2 Bit 5	**5 o o 6**	GND
Port 2 Bit 4	**7 o o 8**	GND
Port 2 Bit 3	**9 o o 10**	GND
Port 2 Bit 2	**11 o o 12**	GND
Port 2 Bit 1	**13 o o 14**	GND
Port 2 Bit 0	**15 o o 16**	GND
Port 1 Bit 7	**17 o o 18**	GND
Port 1 Bit 6	**19 o o 20**	GND
Port 1 Bit 5	**21 o o 22**	GND
Port 1 Bit 4	**23 o o 24**	GND
Port 1 Bit 3	**25 o o 26**	GND
Port 1 Bit 2	**27 o o 28**	GND
Port 1 Bit 1	**29 o o 30**	GND
Port 1 Bit 0	**31 o o 32**	GND
Port 0 Bit 7	**33 o o 34**	GND
Port 0 Bit 6	**35 o o 36**	GND
Port 0 Bit 5	**37 o o 38**	GND
Port 0 Bit 4	**39 o o 40**	GND
Port 0 Bit 3	**41 o o 42**	GND
Port 0 Bit 2	**43 o o 44**	GND
Port 0 Bit 1	**45 o o 46**	GND
Port 0 Bit 0	**47 o o 48**	GND
+5V	**49 o o 50**	GND

J2

Port 5 Bit 7	**1 o o 2**	GND
Port 5 Bit 6	**3 o o 4**	GND
Port 5 Bit 5	**5 o o 6**	GND
Port 5 Bit 4	**7 o o 8**	GND
Port 5 Bit 3	**9 o o 10**	GND
Port 5 Bit 2	**11 o o 12**	GND
Port 5 Bit 1	**13 o o 14**	GND
Port 5 Bit 0	**15 o o 16**	GND
Port 4 Bit 7	**17 o o 18**	GND
Port 4 Bit 6	**19 o o 20**	GND
Port 4 Bit 5	**21 o o 22**	GND
Port 4 Bit 4	**23 o o 24**	GND
Port 4 Bit 3	**25 o o 26**	GND
Port 4 Bit 2	**27 o o 28**	GND
Port 4 Bit 1	**29 o o 30**	GND
Port 4 Bit 0	**31 o o 32**	GND
Port 3 Bit 7	**33 o o 34**	GND
Port 3 Bit 6	**35 o o 36**	GND
Port 3 Bit 5	**37 o o 38**	GND
Port 3 Bit 4	**39 o o 40**	GND
Port 3 Bit 3	**41 o o 42**	GND
Port 3 Bit 2	**43 o o 44**	GND
Port 3 Bit 1	**45 o o 46**	GND
Port 3 Bit 0	**47 o o 48**	GND
+5V	**49 o o 50**	GND

U1 = BASE_ADDR+10H

J3

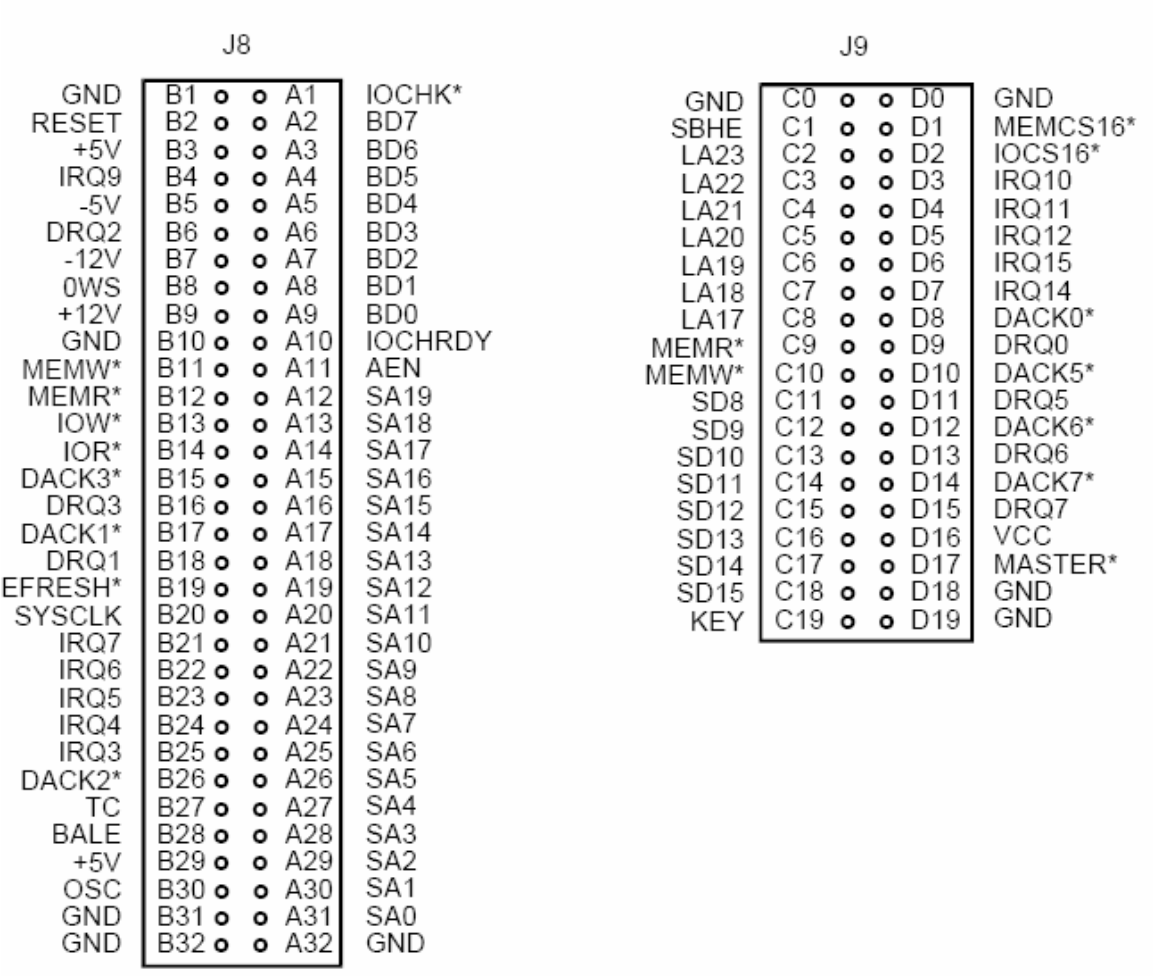
Port 5 Bit 7	**1 o o 2**	GND
Port 5 Bit 6	**3 o o 4**	GND
Port 5 Bit 5	**5 o o 6**	GND
Port 5 Bit 4	**7 o o 8**	GND
Port 5 Bit 3	**9 o o 10**	GND
Port 5 Bit 2	**11 o o 12**	GND
Port 5 Bit 1	**13 o o 14**	GND
Port 5 Bit 0	**15 o o 16**	GND
Port 4 Bit 7	**17 o o 18**	GND
Port 4 Bit 6	**19 o o 20**	GND
Port 4 Bit 5	**21 o o 22**	GND
Port 4 Bit 4	**23 o o 24**	GND
Port 4 Bit 3	**25 o o 26**	GND
Port 4 Bit 2	**27 o o 28**	GND
Port 4 Bit 1	**29 o o 30**	GND
Port 4 Bit 0	**31 o o 32**	GND
Port 3 Bit 7	**33 o o 34**	GND
Port 3 Bit 6	**35 o o 36**	GND
Port 3 Bit 5	**37 o o 38**	GND
Port 3 Bit 4	**39 o o 40**	GND
Port 3 Bit 3	**41 o o 42**	GND
Port 3 Bit 2	**43 o o 44**	GND
Port 3 Bit 1	**45 o o 46**	GND
Port 3 Bit 0	**47 o o 48**	GND
+5V	**49 o o 50**	GND

J4

Port 2 Bit 7	**1 o o 2**	GND
Port 2 Bit 6	**3 o o 4**	GND
Port 2 Bit 5	**5 o o 6**	GND
Port 2 Bit 4	**7 o o 8**	GND
Port 2 Bit 3	**9 o o 10**	GND
Port 2 Bit 2	**11 o o 12**	GND
Port 2 Bit 1	**13 o o 14**	GND
Port 2 Bit 0	**15 o o 16**	GND
Port 1 Bit 7	**17 o o 18**	GND
Port 1 Bit 6	**19 o o 20**	GND
Port 1 Bit 5	**21 o o 22**	GND
Port 1 Bit 4	**23 o o 24**	GND
Port 1 Bit 3	**25 o o 26**	GND
Port 1 Bit 2	**27 o o 28**	GND
Port 1 Bit 1	**29 o o 30**	GND
Port 1 Bit 0	**31 o o 32**	GND
Port 0 Bit 7	**33 o o 34**	GND
Port 0 Bit 6	**35 o o 36**	GND
Port 0 Bit 5	**37 o o 38**	GND
Port 0 Bit 4	**39 o o 40**	GND
Port 0 Bit 3	**41 o o 42**	GND
Port 0 Bit 2	**43 o o 44**	GND
Port 0 Bit 1	**45 o o 46**	GND
Port 0 Bit 0	**47 o o 48**	GND
+5V	**49 o o 50**	GND

2.5 PC/104 BUS INTERFACE

The PCM-UIO96B-G connects to the processor through the PC/104 bus connector at J8 and J9. The pin definitions for J8 and J9 are shown here:



2.6 WS16C48 REGISTER DEFINITIONS

The PCM-UIO96B-G uses two of the WinSystems' exclusive ASIC device, the WS16C48. This device provides 96 lines of digital I/O. There are 17 unique registers within each WS16C48. The following table summarizes the registers. The text that follows provides details on each of the internal registers.

I/O Address Offset	Page 0	Page 1	Page 2	Page 3
00H	Port 0 I/O	Port 0 I/O	Port 0 I/O	Port 0 I/O
01H	Port 1 I/O	Port 1 I/O	Port 1 I/O	Port 1 I/O
02H	Port 2 I/O	Port 2 I/O	Port 2 I/O	Port 2 I/O
03H	Port 3 I/O	Port 3 I/O	Port 3 I/O	Port 3 I/O
04H	Port 4 I/O	Port 4 I/O	Port 4 I/O	Port 4 I/O
05H	Port 5 I/O	Port 5 I/O	Port 5 I/O	Port 5 I/O
06H	INT_PENDING	INT_PENDING	INT_PENDING	INT_PENDING
07H	Page/Lock	Page/Lock	Page/Lock	Page/Lock
08H	N/A	POL_0	ENAB_0	INT_ID0
09H	N/A	POL_1	ENAB_1	INT_ID1
0AH	N/A	POL_2	ENAB_2	INT_ID2

Register Details

Port 0-5 I/O - Each I/O bit in each of these 6 ports can be individually programmed for input or output. Writing a '0' to a bit position causes the corresponding output pin to go to a High Impedance state (pulled high by external 10K ohm resistors). This allows it to be used as an input. When used in the input mode, a read reflects the inverted state of the I/O pin, such that a high on the pin will read as a '0' in the register. Writing a '1' to a bit position causes the output pin to sink current (up to 12mA), effectively pulling it low.

INT_PENDING - This read only register reflects the combined state of the INT_ID0 through INT_ID2 registers. When any of the lower 3 bits are set, it indicates that an interrupt is pending on the I/O port corresponding to the bit position(s) that are set. Reading this register allows an Interrupt Service Routine to quickly determine if any interrupts are pending and which I/O port has an interrupt pending.

PAGE/LOCK – This register serves two purposes. The upper two bits select the register page in use as shown here:

D7	D6	Page
0	0	Page 0
0	1	Page 1
1	0	Page 2
1	1	Page 3

Bits 5-0 allow for locking of the I/O ports. A '1' written to the I/O port position will prohibit further writes to the corresponding I/O port.

POL0 – POL3 – These registers are accessible when page 1 is selected. They allow interrupt polarity selection on a port-by-port and bit-by-bit basis. Writing a '1' to a bit position selects rising edge detection interrupts while writing a '0' to a bit position selects falling edge detection interrupts.

ENAB0 – ENAB3 – These registers are accessible when page 2 is selected. They allow for port-by-port and bit-by-bit enabling of the edge detection interrupts. When set to a '1' the edge detection interrupt is enabled for the corresponding port and bit. When cleared to a '0' the bit's edge detection interrupt is disabled. Note that the register can be used to individually clear a pending interrupt by disabling and re-enabling the pending interrupt.

INT_ID0 – INT_ID2– These registers are accessible when page 3 is selected. They are used to identify currently pending edge interrupts. A bit when read as a '1' indicates that an edge of the polarity programmed into the corresponding polarity register has been recognized. Note that a write to this register (value ignored) clears ALL of the pending interrupts in this register.

2.7 CONNECTOR/JUMPER SUMMARY

CONNECTOR/JUMPER	PURPOSE	PAGE REFERENCE
J1	Ports 0-2 I/O Connector	2-3
J2	Ports 3-5 I/O Connector	2-3
J3	Ports 9-11 I/O Connector	2-3
J4	Ports 6-8 I/O Connector	2-3
J5	Base I/O Address select jumper	2-1
J6	Shared interrupt select jumper	2-2
J7	Interrupt routing jumper	2-2
J8	PC/104-8 bus connector	2-4
J9	PC/104-16 bus connector	2-4

3 PCM-UIO96B-G PROGRAMMING REFERENCE

3.1 INTRODUCTION

This section provides basic documentation for the included I/O routines. It is intended that the accompanying source code equip the programmer with a basic library of I/O functions for the PCM-UIO96B-G or can serve as the basis from which application-specific code can be derived.

The sample I/O routines and sample programs were compiled and tested using the Borland C/C++ compiler Version 3.1. The routines should readily port to any compiler supporting basic port I/O instructions.

These routines handle each WS16C48 as a separate "Chip Number". The PCM-UIO96B-G has two chips 1 and 2.

3.2 FUNCTION DEFINITIONS

This section briefly describes each of the functions contained in the driver. When necessary, short examples will be provided to illustrate usage. Any application making use of any of the driver functions should include the header file **UIO48.H**, which includes the function prototypes and the needed constant definitions.

Note that all of the functions utilize the concept of *bit_number*. The bit_number is a value from 1 to 48 (1 to 24 for interrupt related functions) that correlates to a specific I/O pin. Bit_number 1 is port 0 bit 0 and continues through to bit_number 48 at port 5 bit 7.

INIT_IO – Initializes I/O, set all ports to input

Syntax

void init_io(int chip_number, unsigned io_address);

Description

This function takes two arguments:

chip_number – The chip number 1-4.

io_address – The I/O address of the WS16C48 chip.

There is no return value. This function initializes all I/O pins for input (sets them high), disables all interrupt settings, and sets the image values.

READ_BIT – Reads an I/O port Bit

Syntax

int read_bit(int chip_number, int bit_number);

Description

This function takes two arguments:

chip_number – The chip number 1-4.

bit_number – a value from 1 to 48 that indicates the I/O pin to read from.

This function returns the state of the I/O pin. A **1** is returned if the I/O pin is low and a **0** is returned if the pin is high.

WRITE_BIT – Writes a **1** or **0** to an I/O Pin

Syntax

void write_bit(int chip_number, int bit_number, int value);

Description

This function takes three arguments:

chip_number – The chip number 1-4.

bit_number – a value from 1 to 48 specifying the I/O pin to be acted upon.

value - is ether 1 or 0.

This function allows for writing of a single bit to either a **0** or a **1** as specified by the second argument. There is no return value and other bits in the I/O port are not affected.

SET_BIT – Sets the specified I/O Bit Syntax

Syntax

void set_bit(int chip_number, int bit_number);

Description

This function takes two arguments:

chip_number – The chip number 1-4.

bit_number – a value from 1 to 48 specifying the I/O pin to be set.

This function sets the specified I/O port bit. Note that setting a bit results in the I/O pin actually going low. There is no return value and other bits in the same I/O port are unaffected.

CLR_BIT – Clears the specified I/O Bit

Syntax

void clr_bit(int chip_number, int bit_number);

Description

This function takes two arguments:

chip_number – The chip number 1-4.

bit_number – a value from 1 to 48 specifying the I/O pin to clear.

This function clears the specified I/O bit. Note that clearing the I/O bit results in the actual I/O pin going high. This function does not affect any bits other than the one specified.

ENAB_INT – Enables Edge Interrupt, Select Polarity

Syntax

void enab_int(int chip_number, int bit_number, int polarity);

Description

This function takes three arguments:

chip_number – The chip number 1-4.

bit_number – a value from 1 to 24, specifying the appropriate bit.

polarity - specifies rising or falling edge polarity detect. The constraints RISING and FALLING are defined **UIO48.H**.

This function enables the edge detection circuitry for the specified bit at the specified polarity. It does not unmask the interrupt controller, install vectors, or handle interrupts when they occur. There is no return value and only the specified bit is affected.

DISAB_INT – Disables Edge Detect Interrupt Detection

Syntax

void disab_int(int bit_number);

Description

This function takes two arguments:

chip_number – The chip number 1-4.

bit_number – a value from 1 to 24 specifying the appropriate bit.

This function shuts down the edge detection interrupts for the specified bit. There is no return value and no harm is done by calling this function for a bit which did not have edge detection interrupts enabled. There is no effect on any other bits.

CLR_INT – Clears the specified pending interrupt

Syntax

void clr_int(int chip_number, int bit_number);

Description

This function takes two arguments:

chip_number – The chip number 1-4.

bit_number – a value from 1 to 24 specifying the bit number to reset the interrupt.

This function clears a pending interrupt on the specified bit. It does this by disabling and reenabling the interrupt. The net result after the call is that the interrupt is no longer pending and is renamed for the next transition of the same polarity. Calling this function on a bit that has not been enabled for interrupts will result in its interrupt being enabled with an undefined polarity. Calling this function with no interrupt pending will have no adverse effect. Only the specified bit is affected.

GET_INT – Retrieves bit number of pending interrupt Syntax

Syntax

int get_int(int chip_number);

Description

This function takes one argument:

chip_number – The chip number 1-4.

This function returns either a **0** for no bit interrupts pending or a value between 1 and 24 representing a bit number that has a pending edge detect interrupt. The routine returns with the first interrupt found and begins its search at Port 0 Bit 0 proceeding through to Port 2 Bit 7. It is necessary to use either clr_int() or disab_int() to avoid returning the same bit continuously. This function may either be used in an application's ISR or can be used in the foreground to poll for bit transitions.

3.3 SAMPLE PROGRAMS

There are three sample programs in source code form included on the PCM-UIO96B-G diskette. These programs are not useful by themselves but are provided to illustrate the usage of the I/O functions provided in UIO96C.

FLASH.C

This program was compiled with the Borland C/C++ version 3.1 on the command line with:

bcc flash.c uio96.c

This program illustrates the most basic usage of the PCM-UIO96B-G board. It uses three functions from the driver code. The **init_io()** function is used to initialize the I/O functions and the **set_bit()** and **clr_bit()** functions are used to sequence through all 96 bits turning each on and then off in turn.

POLL.C

This program was compiled with the Borland C/C++ version 3.1 on the command line with:

bcc poll.c uio96.c

This program illustrates the additional features of the WS16C48 and the I/O library functions It programs the first 24 bits for input on each chip, arms them for falling edge detection and then polls the I/O routine **get_init()** to determine if any transitions have taken place.

INT.C

This program was compiled with the Borland C/C++ version 3.1 on the command line with:

bcc int.c uio96.c

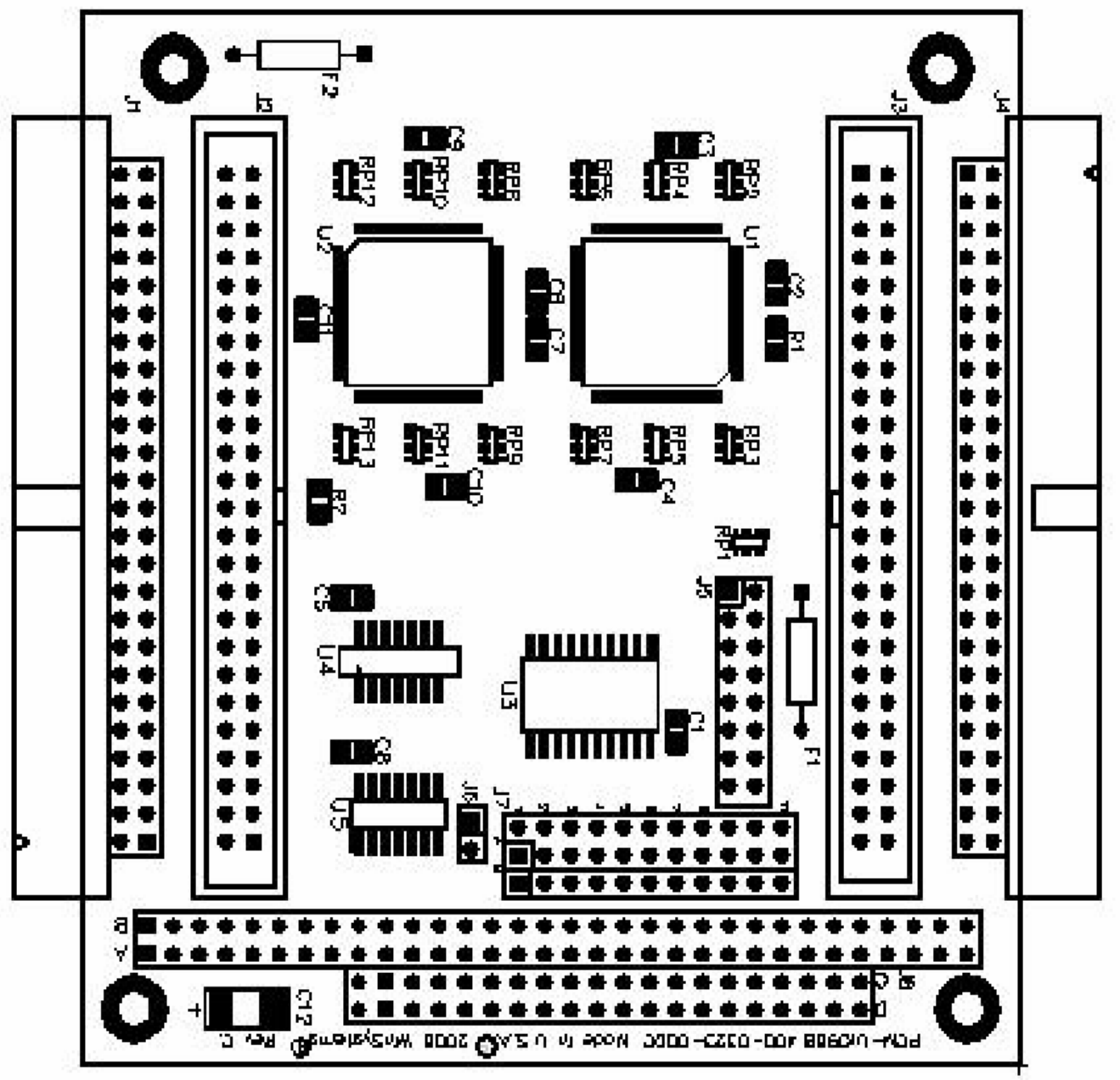
This program is identical in function to the “poll.c” program except that interrupts are active and all updating of the transition counters is accomplished in the background during the interrupt service routine.

Summary

The source code for all three sample programs, as well as the I/O routines, are included on the accompanying diskette. The source code is also provided in printed form in Appendix C. These I/O routines along with the sample programs should provide a good basis on which to build an application utilizing the features of the PCM-UIO96B-G.

APPENDIX A

PCM-UIO96B-G Parts Placement Guide



APPENDIX B

I/O Routine & Sample Program Source Listings

```
/* FLASH.C

        Copyright 1999 by WinSystems Inc.

       Permission is hereby granted to the purchaser of the WinSystems
       UIO cards and CPU products incorporating the UIO device, to distribute
       any binary file or files compiled using this source code directly or
       in any work derived by the user from this file. In no case may the
       source code, original or derived from this file, be distributed to any
       third party except by explicit permission of WinSystems. This file is
       distributed on an 'As-is' basis and no warranty as to performance,
       fitness of purposes, or any other warranty is expressed or implied.
       In no case shall WinSystems be liable for any direct or indirect loss
       or damage, real or consequential resulting from the usage of this
       source code. It is the user's sole responsibility to determine
       fitness for any considered purpose.
*/

#include <stdio . h>
#include <conio . h>
#include <dos.h>
#include 'uio96 .h'

/* This is where we have our board jumpered to */

#define BASE_PORT 0x200


/* This is an utlra-simple demonstration program of some of the functions
  available in the UIO96 source code library. This program simply sets and
  clears each I/O line in succession. It was tested by hooking LEDs to all
  of the I/O lines and wathching the lit one race through the bits.
*/

void main()
{
int x;


       /* Initialize all I/O bits, and set then for input */

        init_io (1 ,BASE_PORT);
        init_io (2 ,BASE_PORT+16);

       /* We'll repeat our sequencing until a key is pressed */

       while(!kbhit())
       {
               /* We will light the LED attached to each of the 48 lines */
               for(x=1; x <=48; x++)
               {
                       /* Setting the bit lights the LED */
                         set_bit(1,x);
                         /* The wait time is subjective. We liked 100mS */
                       delay (100);
                       /* Now turn off the LED */
                         clr_bit(1,x);
               }
                 /* Now move on to the second chip and repeat the pattern */
               for(x=1; x <=48; x++)
               {
                       /* Setting the bit lights the LED */
                         set_bit(2,x);
                         /* The wait time is subjective. We liked 100mS */
                       delay (100);
                       /* Now turn off the LED */
                         clr_bit(2,x);
}

       }
       getch();
```

```
/* INTS.C

        Copyright 1996-1999 by WinSystems Inc.

       Permission is hereby granted to the purchaser of the WinSystems
       UIO cards and CPU products incorporating the UIO device, to distribute
       any binary file or files compiled using this source code directly or
       in any work derived by the user from this file. In no case may the
       source code, original or derived from this file, be distributed to any
       third party except by explicit permission of WinSystems. This file is
       distributed on an 'As-is' basis and no warranty as to performance,
       fitness of purposes, or any other warranty is expressed or implied.
       In no case shall WinSystems be liable for any direct or indirect loss
       or damage, real or consequential resulting from the usage of this
       source code. It is the user's sole responsibility to determine
       fitness for any considered purpose.

*/

#include <stdio . h>
#include <dos.h>
#include <conio . h>
#include 'uio96 .h'

#define BASE_PORT 0x200

/* This program like the poll.c sample uses the edge detection interrupt
  capability of the WS16C48 to count edge transitions. Unlike poll.c,
  however this program actually uses interrupts and update all of the
  transition counters in the background.

*/
/* Our transition totals are stored in this global array */ unsigned
int_counts[2] [25];

/* Function declarations for local functions */

void interrupt int_handler (void);
void interrupt (*old_handler) (void);

void main()
{
int x;

       /* Initialize the I/O ports. Set all I/O pins to input */

        init_io (1 ,BASE_PORT);
        init_io(2,BASE_PORT+16); /* Initialize second chip */

       /* Install an interrupt handler for the board */


       /* We disable interrupts whenever we're changing the environment */

       disable();      /* Disable interrupts during initialization */

       /* Get the old handler and save it for later resoration */

       old_handler = getvect(0x0d); /* Hardwired for IRQ5 */

    /* Install out new interrupt handler */

       setvect (0x0d, int_handler);

       /* Clear the transition count values and enable the falling edge
          interrupts.
       */
       for(x=1; x<25; x++)
       {
```

```
                int _counts [0] [x] = 0;    /* Clear the counts */
                int _counts [1] [x] = 0;
                enab_int(1,x,FALLING);      /* Enable the falling edge interrupts */
                enab_int (2, x, FALLING);
        }


        /* Unmask the interrupt controller */
        outportb(0x21,(inportb(0x21) & 0xdf));        /* Unmask IRQ 5 */

        /* Reenable interrupts */
        enable();


        /* Set up the display */
        clrscr();       /* Clear the Text Screen */

        for(x=1; x<25; x++)
        {
               gotoxy(1,x);
               printf('Bit Number %02d ',x);
                 gotoxy(39,x);
                printf('Bit Number %02d ',x);
        }


        /* We will continuously print the transition totals until a
           key is pressed */


        /* All of the processing of the transition interrupts, including
           updating the counts is done in the background when an interrupt
           occurs.
        */

        while(!kbhit())
        {
               for(x=1; x < 25; x++)
               {
                        gotoxy(16,x);
                        printf('%05u',int_counts[0] [x]);

                        gotoxy(55,x);
                        printf('%05u',int_counts[1] [x]);

}
        }

        getch();
        /* Disable interrupts while we restore things */

        disable();

        /* Mask off the interrupt at the interrupt controller */

        outportb(0x21,inportb(0x21) |  0x20); /* Mask IRQ 5 */


        /* Restore the old handler */

        setvect(0x0d,old_handler);      /* Put back the old interrupt handler */

        /* Reenable interrupts. Things are back they way they were before we
           started.
*/

        enable();
}

/* This function is executed when an edge detection interrupt occurs */

void interrupt int_handler(void)
{
```

```
int current1 , current2;

/* Get the current interrupt pending. There really should be one here or
  we shouldn't even be executing this function.
    */

        while(1)        /* stay here till all
        handled */
        {
        current1 = get _int(1);

        /* We will continue processing pending edge detect interrupts until
        there are no more present. In which case current == 0
        */

        if (current1)
        {
        clr_int (1, current1);

        /* Tally up one for the current bit number */

        ++int _counts [0] [current1];
        }


        /* Get the next one, if any others pending */ current2

        = get_int(2);

        if (current2)
        {
        clr_int (2, current2);
        ++int _counts [1] [current2];
        }

        if((current1 == 0) && (current2 == 0))
        {
        current1 = get_int(1); current2 =
        get_int(2);
        if((current1 == 0) && (current2 == 0))
        break;
        }
        }

        /* Issue a non-specific end of interrupt command (EOI) to the
        interrupt controller. This rearms it for the next shot.
        */

        outportb(0x20,0x20); /* Do non-specific EOI */
}
```

```
/* POLL.C

        Copyright 1999 by WinSystems Inc.

       Permission is hereby granted to the purchaser of the WinSystems
       UIO cards and CPU products incorporating the UIO device, to distribute
       any binary file or files compiled using this source code directly or
       in any work derived by the user from this file. In no case may the
       source code, original or derived from this file, be distributed to any
       third party except by explicit permission of WinSystems. This file is
       distributed on an 'As-is' basis and no warranty as to performance,
       fitness of purposes, or any other warranty is expressed or implied.
       In no case shall WinSystems be liable for any direct or indirect loss
       or damage, real or consequential resulting from the usage of this
       source code. It is the user's sole responsibility to determine
       fitness for any considered purpose.

*/

#include <stdio . h>
#include <conio . h>
#include 'uio96 .h'


#define BASE_PORT 0x200

/* This program uses the edge detection interrupt capability of the
  WS16C48 to count transitions on the first 24 lines of each chip.
  It does this however, not by using true interrupts, but by polling
  for transitions using the get_int() function.

*/


/* Our transition totals are stored in this array */


unsigned int_counts[2] [25];

/* Definitions for local functions */

void check_ints (void);

void main()
{
int x;


       /* Initialize the I/O ports. Set all I/O pins to input */

        init_io (1 ,BASE_PORT);
        init_io(2,BASE_PORT+16); /* Initialize the second chip as well */


       /* Initialize our transition counts, and enable falling edge
          transition interrupts.
       */

       for(x=1; x<25; x++)
       {
                int _counts [0] [x] = 0;    /* Clear the counts */
                int _counts [1] [x] = 0;
                enab_int(1,x,FALLING);      /* Enable the falling edge interrupts */
                enab_int (2, x, FALLING);
       }

       /* Clean up the screen for our display. Nothing fancy */
       clrscr();

       for(x=1; x<25; x++)
       {

               gotoxy(1,x);
```

```
                printf('Bit number %02d ',x);
                  gotoxy(39,x);
                printf('Bit number %02d ',x);
        }


       /* We will continue to display until any key is pressed */

        while(!kbhit())
        {

                /* Retrieve any pending transitions and update the counts */

                check_ints();

                /* Display the current count values */

                for(x=1; x < 25; x++)
                {
                        gotoxy(16,x);
                          printf('%05u',int_counts[0] [x]);
                          gotoxy(55,x);
                          printf('%05u',int_counts[1] [x]);
}
        }
       getch();
}

void check_ints()
{
int current;

        /* Get the bit number of a pending transition interrupt */

        current = get_int(1);

       /* If it's 0 there are none pending */

        if (current != 0)
        {
                /* Clear and rearm this one so we can get it again */

                clr_int (1, current);

                /* Tally a transition for this bit */

                ++int _counts [0] [current];
        }

        current = get_int(2);

        /* If it's 0 there are none pending */

        if (current != 0)
        {
                /* Clear and rearm this one so we can get it again */

                clr_int (2, current);
                /* Tally a transition for this bit */

                ++int _counts [1] [current];

}
```

```
/* UIO96.C

        Copyright 1996-1999 by WinSystems Inc.

       Permission is hereby granted to the purchaser of the WinSystems
       UIO cards and CPU products incorporating the UIO device, to distribute
       any binary file or files compiled using this source code directly or
        in any work derived by the user from this file. In no case may the
        source code, original or derived from this file, be distributed to any
       third party except by explicit permission of WinSystems. This file is
       distributed on an 'As-is' basis and no warranty as to performance,
        fitness of purposes, or any other warranty is expressed or implied.
        In no case shall WinSystems be liable for any direct or indirect loss
       or damage, real or consequential resulting from the usage of this
        source code. It is the user's sole responsibility to determine
        fitness for any considered purpose.
*/
/*****************************************************************************
*       Name     :   u i o 9 6 . c
*
*        Project : PCM-UIO96 Software Samples/Examples
*
*        Date    : August 20, 1993
*
*       Revision: 1.00
*
*       Author : Steve Mottin
*
*****************************************************************************
*       Changes :
*
*       Date            Revision       Description
*       _________       _________      ____________________________________________

*        08/20/991.00    Adapted from UIO48 code
*
*****************************************************************************
*/

#include <dos.h>

/* This global holds the base address of the UIO chip */

unsigned base_port[4] = {0,0,0,0};

/* This global array holds the image values of the last write to each I/O
  ports. This allows bit manipulation routines to work without having to
  actually do a read-modify-write to the I/O port.
*/

unsigned port_images [4] [6];

/*===========================================================================
*                               INIT _IO
*
*  This function take two arguments :
*
*
*  Chip_number :Chip number 1 to 4;
*  io_address :This is the base I/O address of the 16C48 UIO Chip
*                                     on the board.
*
*
*  This function initializes all I/O pins for input, disables all interrupt
*  sensing, and sets the image values.
*
*===========================================================================*/

void init_io(int chip_number,unsigned io_address)
{
int x;
```

```
        /* Zero adjust chip number */

        – -chip_number;

        /* Save the specified address for later use */

        base_port[chip_number] = io_address;

        /* Clear all of the I/O ports. This also makes them inputs */

        for(x=0; x < 7; x++)
                outportb(base_port[chip_number]+x, 0);

        /* Clear our image values as well */

        for(x=0; x < 6; x++)
                port_images [chip_number] [x] = 0;

        /* Set page 2 access, for interrupt enables */

        outportb (base_port [chip_number] +7, 0x80);

        /* Clear all interrupt enables */

        outportb (base_port [chip_number] +8,0);
        outportb (base_port [chip_number] +9,0);
        outportb (base_port [chip_number] +0x0a, 0);

        /* Restore normal page 0 register access */
        outportb (base_port [chip_number] +7,0);

}

/*=========================================================================
*                                READ _BIT
*
*
*  This function takes two arguments :
*
*  chip_number     : This argument specifies the chip number 1 to 4
*
*  bit_number      : The integer argument specifies the bit number to read.
*                    Valid arguments are from 1 to 48.
*
*  return value : The current state of the specified bit, 1 or 0.
*
*  This function returns the state of the current I/O pin specified by
*  the argument bit_number.
*=========================================================================*/

int read_bit(int chip_number,int bit_number)
{
unsigned port;
int val;

        /* Adjust chip number to zero based addressing */

        – -chip_number;

        /* Adjust the bit_number to 0 to 47 numbering */

        - -bit_number;

        /* Calculate the I/O port address based on the updated bit_number */

        port = (bit _number / 8) + base_port[chip _number];

        /* Get the current contents of the port */

       val = inportb(port);
```

```
        /* Get just the bit we specified */

        val = val & (1 << (bit _number % 8));

        /* Adjust the return for a 0 or 1 value */

        if (val)
                return 1;

        return 0;
}

/*
*=============================================================================
*
*                                WRITE_BIT
*
* This function takes three arguments :
*
*
*  chip_number : The I/O Chip number 1 to 4
*
*  bit_number : The I/O pin to access is specified by bit_number 1 to 48.
*
*  val :     The setting for the specified bit, either 1 or 0.
*
*  This function sets the specified I/O pin to either high or low as dictated
*  by the val argument. A non zero value for val sets the bit.
*
*============================================================================*/

void write_bit(int chip_number, int bit_number, int val)
{
unsigned  port;
unsigned  temp;
unsigned mask;

        /* Adjust chip number for 0 based numbering */

        - -chip_number;

        /* Adjust bit_number for 0 based numbering */

        - -bit_number;

        /* Calculate the I/O address of the port based on the bit number */

        port = (bit _number / 8) + base_port[chip _number];

        /* Use the image value to avoid having to read the port first. */

         temp = port_images [chip_number] [bit_number / 8];    /* Get current value */

        /* Calculate a bit mask for the specified bit */

        mask = (1 << (bit _number % 8));

        /* Check whether the request was to set or clear and mask accordingly */

        if(val) /* If the bit is to be set */
                temp = temp | mask;
        else
                temp = temp & ~mask;

        /* Update the image value with the value we're about to write */

        port_images [chip_number] [bit_number / 8] = temp;

        /* Now actually update the port. Only the specified bit is affected */

        outportb (port, temp);
```

```
/*=========================================================================
*                                  SET _BIT
*
*
*  This function takes two arguments :
*
*  chip_number : The chip number from 1 to 3
*
*  bit_number : The bit number to set 1 to 48.
*
*  This function sets the specified bit.
*
*=========================================================================*/

void set_bit(int chip_number,int bit_number)
{

       write_bit (chip_number ,bit_number, 1);
}
/*=========================================================================
*                                  CLR _BIT
*
*
*  This function takes two arguments :
*
*  chip_number: Chip number 1 to 3
*
*  bit_number : The bit number to clear.
*
*  This function clears the specified bit.
*
*=========================================================================*/

void clr_bit(int chip_number,int bit_number)
{
       write_bit (chip_number ,bit_number, 0);
}
/*=========================================================================
*
*                                ENAB _INT
*
*  This function takes three arguments :
*
*  chip_number : The desired chip number 1 to 4
*
*  bit_number : The bit number to enable intterups for. Range from 1 to 48.
*
*  polarity    : This specifies the polarity of the interrupt. A non-zero
*                argument enables rising-edge interrupt. A zero argument
*                enables the interrupt on the flling edge.
*
*  This function enables within the 16C48 an interrupt for the specified bit
*  at the specified polarity. This function does not setup the interrupt
*  controller, nor does it supply an interrupr handler.
*
*=========================================================================*/

void enab_int(int chip_number,int bit_number, int polarity)
{
unsigned   port;
unsigned   temp;
unsigned mask;

       /* Adjust chip number for 0 based numbering */

       - -chip_number;

       /* Adjust for 0 based numbering */
```

```
        - -bit_number;

        /* Calculate the I/O address based uppon the bit number */

        port = (bit_number / 8) + base_port[chip_number] + 8;

        /* Calculate a bit mask based on the specified bit number */

        mask = (1 << (bit_number % 8));

        /* Turn on page 2 access */

        outportb (base_port [chip_number] +7, 0x80);

        /* Get the current state of the interrupt enable register */

        temp = inportb (port);

        /* Set the enable bit for our bit number */

        temp = temp | mask;

        /* Now update the interrupt enable register */

        outportb (port, temp);

        /* Turn on access to page 1 for polarity control */

        outportb (base_port [chip_number] +7, 0x40);

        /* Get the current state of the polarity register */

        temp = inportb(port);           /* Get current polarity settings */

        /* Set the polarity according to the argument in the image value */

        if(polarity)            /* If the bit is to be set */
                temp = temp | mask;
        else
                temp = temp & ~mask;

        /* Write out the new polarity value */

        outportb (port, temp);

        /* Set access back to Page 0 */

        outportb (base_port [chip_number] +7, 0x0);

}

/*
*
*=============================================================================
*                                       DISAB_INT
*
* This function takes two arguments :
*
* chip_number: Desired chip number 1 to 4

* bit_number : Specifies the bit number to act upon. Range is from 1 to 48.
*
* This function shuts off the interrupt enabled for the specified bit.
*
*=============================================================================*/

void disab_int(int chip_number,int bit_number)
{
unsigned  port;
unsigned  temp;
unsigned mask;

        /* Adjust for 0 based addressing */
```

```
    -     -chip_number;

        /* Adjust the bit_number for 0 based numbering */

        - -bit_number;

        /* Calculate the I/O Address for the enable port */

        port = (bit_number / 8) + base_port[chip_number] + 8;

        /* Calculate the proper bit mask for this bit number */

        mask = (1 << (bit_number % 8));

        /* Turn on access to page 2 registers */

        outportb (base_port [chip_number] +7, 0x80);

        /* Get the current state of the enable register */

        temp = inportb (port);

        /* Clear the enable bit int the image for our bit number */

        temp = temp & ~mask;

        /* Update the enable register with the new information */

        outportb (port, temp);

        /* Set access back to page 0 */

        outportb (base_port [chip_number] +7, 0x0);

}

/*
*
*============================================================================
*                                 CLR _INT
*
*  This function takes two arguments :
*
*  chip_number : The desired chip number 1 to 4
*
*  bit_number : This argument specifies the bit interrupt to clear. Range
*               is 1 to 24.
*
*
*  This function is use to clear a bit interrupt once it has been recognized.
*  The interrupt left enabled.
*
*============================================================================*/

void clr_int(int chip_number,int bit_number)
{
unsigned  port;
unsigned  temp;
unsigned mask;

        /* Adjust chip number for 0 based */

         - -chip_number;

        /* Adjust for 0 based numbering */

        - -bit_number;

        /* Calculate the correct I/O address for our enable register */

        port = (bit _number / 8) + base_port[chip _number] + 8;
```

```
        /* Calculate a bit mask for this bit number */ mask =

        (1 << (bit_number % 8));

        /* Set access to page 2 for the enable register */ outportb

        (base_port [chip_number] +7, 0x80); /* Get current

        state of the enable register */ temp = inportb (port);

            /* Temporarily clear only OUR enable. This clears the interrupt */

        temp = temp & ~mask;            /* clear the enable for this bit */

        /* Write out the temporary value */

        outportb (port, temp);

        /* Re-enable our interrupt bit */

        temp = temp | mask;

        /* Write it out */

        outportb (port, temp);

        /* Set access back to page 0 */

        outportb (base_port [chip_number] +7, 0x0);

}


/*
*
*=============================================================================
*                                GET _INT
*
*  This function takes one arguments.
*
*  chip_number : The desired chip number 1 to 4
*
*  return value : The value returned is the highest level bit interrupt
*                 currently pending. Range is 1 to 24.
*
*  This function returns the highest level interrupt pending. If no interrupt
*  is pending, a zero is returned. This function does NOT clear the interrupt.
*============================================================================*/

int get_int(int chip_number) {
int temp;
int x;

        /* Adjust the chip number for zero based numbering */

        - -chip_number;

        /* read the master interrupt pending register, mask off undefined bits */ temp =

        inportb(base_port[chip_number]+6) & 0x07;

        /* If there are no interrupts pending, return a 0 */

        if((temp & 7) == 0) return
                (0);

        /* There is something pending, now we need to identify what it is */
```

```
        /* Set access to page 3 for interrupt id registers */ outportb (base_port [chip_number]
        +7, 0xc0);
        /* Read interrupt ID register for port 0 */

        temp = inportb (base_port [chip_number] +8);


        /* See if any bit set, if so return the bit number */

        if(temp !=0)
        {
               for(x=0; x <=7; x++)
               {
                       if (temp & (1 << x))
                       {
                                  outportb(base_port[chip_number]+7,0);         /* Turn off access
*/
                                 return(x+1);            /* Return bitnumber with active int */
        }
               }
        }

        /* None in Port 0, read port 1 interrupt ID register */

        temp = inportb(base_port[chip_number]+9);

        /* See if any bit set, if so return the bit number */

        if(temp !=0)
        {
               for(x=0; x <=7; x++)
               {
                       if (temp & (1 << x))
                       {
                                  outportb(base_port[chip_number]+7,0);         /* Turn off access
*/
                                 return(x+9);            /* Return bitnumber with active int */
        }
               }
        }

        /* Lastly, read status of port 2 int id */

        temp = inportb(base_port[chip_number]+0x0a);     /* Read port 2 status */

        /* If any pending, return the appropriate bit number */

        if(temp !=0)
        {
               for(x=0; x <=7; x++)
               {
                       if (temp & (1 << x))
                       {
                                  outportb(base_port[chip_number]+7,0);         /* Turn off access
*/
                                 return(x+17);          /* Return bitnumber with active int */
        }
               }
        }

        /* We should never get here unless the hardware is misbehaving but just
           to be sure. We'll turn the page access back to 0 and return a 0 for
           no interrupt found.
        */


        outportb (base_port [chip_number] +7,0);
        return 0;
```

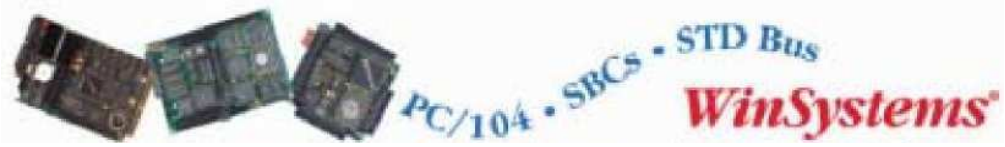
```
/* UIO96.H

        Copyright 1996-1999 by WinSystems Inc.

              Permission is hereby granted to the purchaser of the WinSystems
       UIO cards and CPU products incorporating the UIO device, to distribute any binary
       file or files compiled using this source code directly or in any work derived by
       the user from this file. In no case may the source code, original or derived from
       this file, be distributed to any third party except by explicit permission of
       WinSystems. This file is distributed on an 'As-is' basis and no warranty as to
       performance, fitness of purposes, or any other warranty is expressed or implied.
        In no case shall WinSystems be liable for any direct or indirect loss or damage,
       real or consequential resulting from the usage of this source code. It is the
       user's sole responsibility to determine
        fitness for any considered purpose.
*/
/*************************************************************
*       Name     :   u i o 9 6 . h
*
*        Project : PCM-UIO96 Software Samples/Examples
*
*        Date    : August 19, 1999
*
*       Revision: 1.00
*
*       Author : Steve Mottin
*
*
*       Changes :
*
*       Date            Revision        Description
*
*        10/30/96         1.00             Adpated from UIO48A code
*************************************************************
*/

#define RISING 1 #define
FALLING 0

void init_io(int chip_number,unsigned io_address); int
read_bit(int chip_number,int bit_number); void
write_bit(int chip_number,int bit_number); void set_bit(int
chip_number,int bit_number); void clr_bit(int
chip_number,int bit_number);
void enab_int(int chip_number,int bit_number, int polarity);
void disab_int(int chip_number,int bit_number); void
clr_int(int chip_number,int bit_number); int get_int(int
chip_number);
```



Telephone: 817-274-7553 . . Fax: 817-548-1358 http://www.winsystems.com .
. E-mail: info@winsystems.com

WARRANTY

WinSystems warrants that for a period of two (2) years from the date of shipment any Products and Software purchased or licensed hereunder which have been developed or manufactured by WinSystems shall be free of any material defects and shall perform substantially in accordance with WinSystems' specifications therefore. With respect to any Products or Software purchased or licensed hereunder which have been developed or manufactured by others, WinSystems shall transfer and assign to Customer any warranty of such manufacturer or developer held by WinSystems, provided that the warranty, if any, may be assigned. The sole obligation of WinSystems for any breach of warranty contained herein shall be, at its option, either (i) to repair or replace at its expense any materially defective Products or Software, or (ii) to take back such Products and Software and refund the Customer the purchase price and any license fees paid for the same. Customer shall pay all freight, duty, broker's fees, insurance changes and other fees and charges for the return of any Products or Software to WinSystems under this warranty. WinSystems shall pay freight and insurance charges for any repaired or replaced Products or Software thereafter delivered to Customer within the United States. All fees and costs for shipment outside of the United States shall be paid by Customer. The foregoing warranty shall not apply to any Products or Software which have been subject to abuse, misuse, vandalism, accidents, alteration, neglect, unauthorized repair or improper installations.

THERE ARE NO WARRANTIES BY WINSYSTEMS EXCEPT AS STATED HEREIN. THERE ARE NO OTHER WARRANTIES EXPRESS OR IMPLIED INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE, IN NO EVENT SHALL WINSYSTEMS BE LIABLE FOR CONSEQUENTIAL, INCIDENTAL, OR SPECIAL DAMAGES INCLUDING, BUT NOT LIMITED TO, DAMAGES FOR LOSS OF DATA, PROFITS OR GOODWILL. WINSYSTEMS' MAXIMUM LIABILITY FOR ANY BREACH OF THIS AGREEMENT OR OTHER CLAIM RELATED TO ANY PRODUCTS, SOFTWARE, OR THE SUBJECT MATTER HEREOF, SHALL NOT EXCEED THE PURCHASE PRICE OR LICENSE FEE PAID BY CUSTOMER TO WINSYSTEMS FOR THE PRODUCTS OR SOFTWARE OR PORTION THEREOF TO WHICH SUCH BREACH OR CLAIM PERTAINS.

WARRANTY SERVICE

All products returned to WinSystems must be assigned a Return Material Authorization (RMA) number. To obtain this number, please call or FAX WinSystems' factory in Arlington, Texas and provide the following information:

1. Description and quantity of the product(s) to be returned including its serial number.
2. Reason for the return.
3. Invoice number and date of purchase (if available), and original purchase order number.
4. Name, address, telephone and FAX number of the person making the request.
5. Do not debit WinSystems for the repair. WinSystems does not authorize debits.

After the RMA number is issued, please return the products promptly. Make sure the RMA number is visible on the outside of the shipping package.

The customer must send the product freight prepaid and insured. The product must be enclosed in an anti-static bag to protect it from damage caused by static electricity. Each bag must be completely sealed. Packing material must separate each unit returned and placed as a cushion between the unit(s) and the sides and top of the shipping container. WinSystems is not responsible for any damage to the product due to inadequate packaging or static electricity.

Artisan Technology Group is an independent supplier of quality pre-owned equipment

Gold-standard solutions

Extend the life of your critical industrial, commercial, and military systems with our superior service and support.

We buy equipment

Planning to upgrade your current equipment? Have surplus equipment taking up shelf space? We'll give it a new home.

Learn more!

Visit us at **artisantg.com** for more info on price quotes, drivers, technical specifications, manuals, and documentation.

Artisan Scientific Corporation dba Artisan Technology Group is not an affiliate, representative, or authorized distributor for any manufacturer listed herein.

We're here to make your life easier. How can we help you today?

(217) 352-9330 | sales@artisantg.com | artisantg.com

ARTISAN® TECHNOLOGY GROUP