OPERATIONS MANUAL
PCM-UIO48B

NOTE: *This manual has been designed and created for use as part of the WinSystems Technical Manuals CD and/or the WinSystems website. If this manual or any portion of the manual is downloaded, copied or emailed, the links to additional information (i.e. software, cable drawings) may be inoperable.*

REVISION HISTORY

P/N 403-0321-000

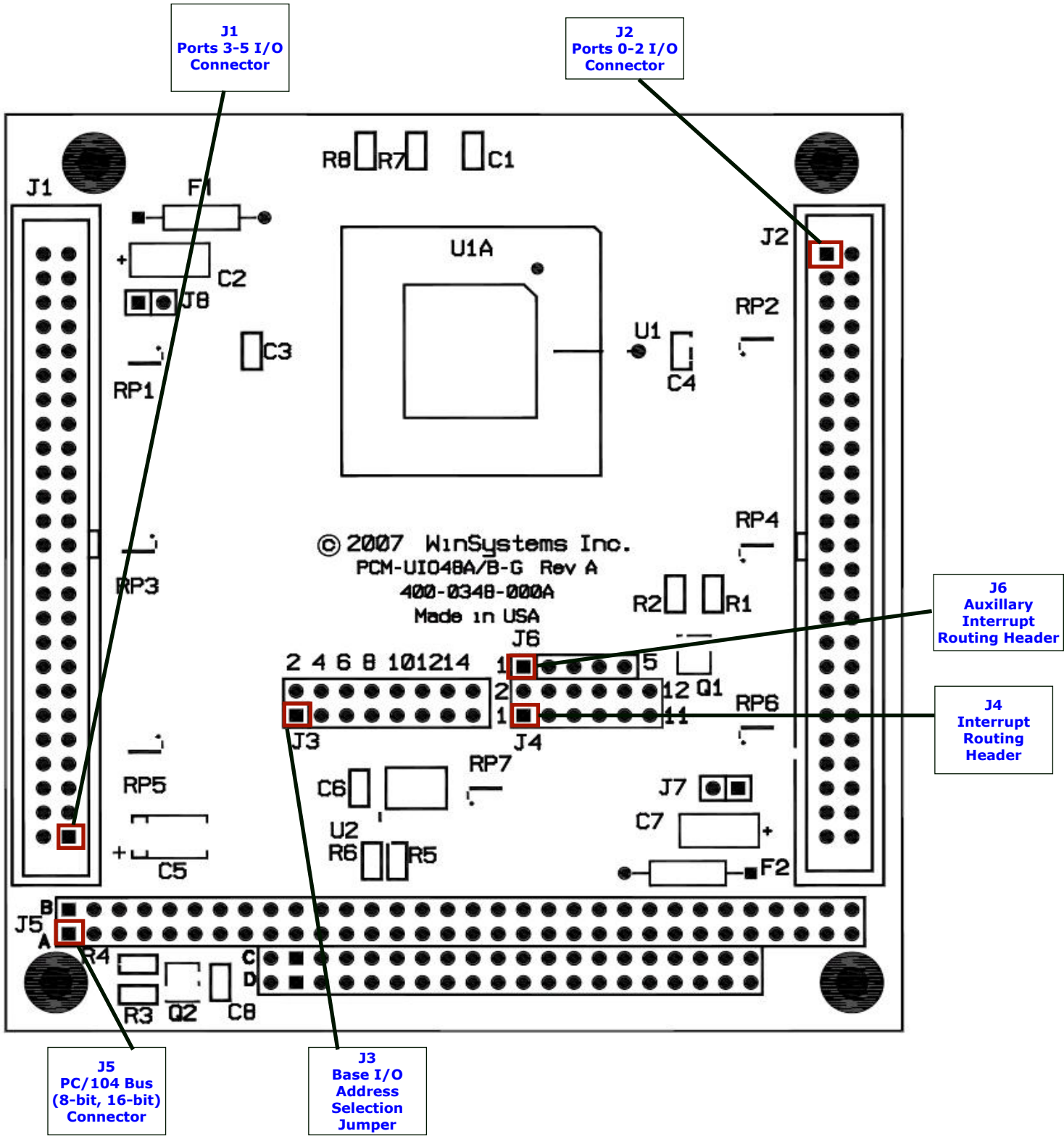
ECO Number	Date Code	Rev Level
ORIGINATED	110425	A

Table of Contents

Visual Index – Quick Reference

Top View - Connectors

For the convenience of the user, a copy of the Visual Index has been provided with direct links to connector and jumper configuration data.



NOTE: The reference line to each component part has been drawn to Pin 1, where applicable. Pin 1 is also highlighted with a red square, where applicable.

Introduction

This manual is intended to provide the necessary information regarding configuration and usage of the PCM-UIO48B board. WinSystems maintains a Technical Support Group to help answer questions regarding usage or programming of the board. For answers to questions not adequately addressed in this manual, contact Technical Support at (817) 274-7553, Monday through Friday, between 8 AM and 5 PM Central Standard Time (CST).

General Information

Features

Digital I/O
- 48 Bidirectional lines with Input, Output or Output with Readback (WS16C48)
- 12 mA Sink Current

Event Sense
- Supports 24 event sense lines
- Programmable polarity for each line
- Software-enabled interrupt for each Line
- Change-of-state latched for each line

Power
- +5V @ 12 mA required

Industrial Operating Temperature Range
- -40°C to 85°C

Form Factor
- PC/104-compliant
- 3.60” x 3.80” (90 mm x 96 mm)

Additional Specifications
- Compatible with industry standard I/O racks
- Write-protection mask register for each port
- Fused +5V logic supply for I/O modules
- 8-bit, 16-bit PC/104 Interface

General Description

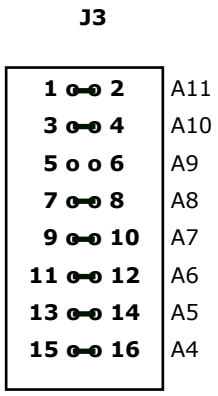
The PCM-UIO48B is a highly versatile PC/104 input/output module providing 48 lines of digital I/O. It is unique in its ability to monitor 24 lines for both rising and falling digital edge transitions, latch them, and then issue an interrupt to the host processor. The application interrupt service routine can quickly determine, through a series of interrupt identification registers, the exact port(s) and bit(s) which have transitioned. The PCM-UIO48B utilizes WinSystems’ WS16C48 ASIC High Density I/O (HDIO) Chip. The first 24 lines are capable of fully latched event sensing with the sense polarity being software programmable. Two 50-pin I/O connectors allow for easy mating with industry standard I/O racks.

Functional Capability

I/O Address Selection

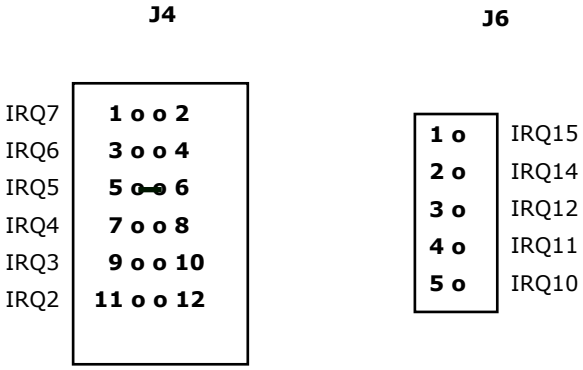
The PCM-UIO48B requires 16 consecutive I/O addresses beginning on an 16-byte boundary. The jumper block at **J3** allows for user selection of the base address. Address selection is made by placing a jumper on the jumper pair for the address bit, if a **0** is desired or leaving the jumper pair open if a **1** is required for the desired address. The illustration below shows the relationship between the address bit and the jumper position and a sample jumpering for an address of 200H.

I/O Base Address Select jumper **J3** shown jumpered for 200H



Interrupt Routing

The PCM-UIO48B can generate an interrupt on up to 24 different lines each with its own polarity select. Interrupt support is provided on the first 24 bits of each device for ports 0, 1 and 2. This interrupt can be routed to the PC/104 bus via the jumper at **J4**. 16-bit versions of the board will also have the auxillary jumper at **J6** installed. The interrupt routing header is shown below along with sample jumpering for IRQ5.



Digital I/O

The PCM-UIO48B routes its 48 lines to 50-pin IDC connectors at **J1** and **J2**. The pin definitions for **J1** and **J2** are shown below.

J2

P2-7	**1 o o 2**	GND
P2-6	**3 o o 4**	GND
P2-5	**5 o o 6**	GND
P2-4	**7 o o 8**	GND
P2-3	**9 o o 10**	GND
P2-2	**11 o o 12**	GND
P2-1	**13 o o 14**	GND
P2-0	**15 o o 16**	GND
P1-7	**17 o o 18**	GND
P1-6	**19 o o 20**	GND
P1-5	**21 o o 22**	GND
P1-4	**23 o o 24**	GND
P1-3	**25 o o 26**	GND
P1-2	**27 o o 28**	GND
P1-1	**29 o o 30**	GND
P1-0	**31 o o 32**	GND
P0-7	**33 o o 34**	GND
P0-6	**35 o o 36**	GND
P0-5	**37 o o 38**	GND
P0-4	**39 o o 40**	GND
P0-3	**41 o o 42**	GND
P0-2	**43 o o 44**	GND
P0-1	**45 o o 46**	GND
P0-0	**47 o o 48**	GND
+5V	**49 o o 50**	GND

J1

P5-7	**1 o o 2**	GND
P5-6	**3 o o 4**	GND
P5-5	**5 o o 6**	GND
P5-4	**7 o o 8**	GND
P5-3	**9 o o 10**	GND
P5-2	**11 o o 12**	GND
P5-1	**13 o o 14**	GND
P5-0	**15 o o 16**	GND
P4-7	**17 o o 18**	GND
P4-6	**19 o o 20**	GND
P4-5	**21 o o 22**	GND
P4-4	**23 o o 24**	GND
P4-3	**25 o o 26**	GND
P4-2	**27 o o 28**	GND
P4-1	**29 o o 30**	GND
P4-0	**31 o o 32**	GND
P3-7	**33 o o 34**	GND
P3-6	**35 o o 36**	GND
P3-5	**37 o o 38**	GND
P3-4	**39 o o 40**	GND
P3-3	**41 o o 42**	GND
P3-2	**43 o o 44**	GND
P3-1	**45 o o 46**	GND
P3-0	**47 o o 48**	GND
+5V	**49 o o 50**	GND

NOTE:
Pin 49 on each connector can supply +5V to the I/O rack. The supply on each connector is protected from excessive current by a 1A miniature fuse F1 for **J1** and F2 for **J2**.

PC/104 Bus Interface

The PCM-UIO48B connects to the processor through the PC/104 bus connector at **J5**. The pin definitions for the 8-bit and 16-bit extension of **J5** are provided here for reference. Refer to the PC/104 Bus Specification for specific signal and mechanical specifications.

GND	**D0 o o C0**	GND
MEMCS16#	**D1 o o C1**	SBHE#
IOCS16#	**D2 o o C2**	LA23
IRQ10	**D3 o o C3**	LA22
IRQ11	**D4 o o C4**	LA21
IRQ12	**D5 o o C5**	LA20
IRQ15	**D6 o o C6**	LA19
IRQ14	**D7 o o C7**	LA18
DACK0#	**D8 o o C8**	LA17
DRQ0	**D9 o o C9**	MEMR#
DACK5#	**D10 o o C10**	MEMW#
DRQ5	**D11 o o C11**	SD8
DACK6#	**D12 o o C12**	SD9
DRQ6	**D13 o o C13**	SD10
DACK7#	**D14 o o C14**	SD11
DRQ7	**D15 o o C15**	SD12
+5V	**D16 o o C16**	SD13
MASTER#	**D17 o o C17**	SD14
GND	**D18 o o C18**	SD15
GND	**D19 o o C19**	KEY

= Active Low Signal

IOCHK#	**A1 o o B1**	GND
SD7	**A2 o o B2**	RESET
SD6	**A3 o o B2**	+5V
SD5	**A4 o o B4**	IRQ9
SD4	**A5 o o B5**	-5V
SD3	**A6 o o B6**	DRQ2
SD2	**A7 o o B7**	-12V
SD1	**A8 o o B8**	SRDY#
SD0	**A9 o o B9**	+12V
IOCHRDY	**A10 o o B10**	KEY
AEN	**A11 o o B11**	SMEMW#
SA19	**A12 o o B12**	SMEMR#
SA18	**A13 o o B13**	IOW#
SA17	**A14 o o B14**	IOR#
SA16	**A15 o o B15**	DACK3#
SA15	**A16 o o B16**	DRQ3
SA14	**A17 o o B17**	DACK1#
SA13	**A18 o o B18**	DRQ1
SA12	**A19 o o B19**	REFRESH#
SA11	**A20 o o B20**	BCLK
SA10	**A21 o o B21**	IRQ7
SA9	**A22 o o B22**	IRQ6
SA8	**A23 o o B23**	IRQ5
SA7	**A24 o o B24**	IRQ4
SA6	**A25 o o B25**	IRQ3
SA5	**A26 o o B26**	DACK2#
SA4	**A27 o o B27**	TC
SA3	**A28 o o B28**	BALE
SA2	**A29 o o B29**	+5V
SA1	**A30 o o B30**	OSC
SA0	**A31 o o B31**	GND
GND	**A32 o o B32**	GND

NOTES:
1. Rows C and D are not required on 8-bit modules.
2. B10 and C19 are key locations. WinSystems uses key pins as connections to GND.
3. Signal timing and function are as specified in ISA specification.
4. Signal source/sink current differ from ISA values.

Software Summary

WS16C48 Register Definitions – The PCM-UIO48B uses the WinSystems exclusive ASIC device, the WS16C48. This device provides 48 lines of digital I/O. There are 17 unique registers within the WS16C48. The following table summarized the registers and 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 through 5 I/O – Each I/O bit in each of the six 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 10 KΩ 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 that output pin to sink current (up to 12 mA), 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 three 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 a pending interrupt.

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 the I/O ports. A **1** written to the I/O port position will prohibit further writes to the corresponding I/O port.

POL0 - POL2 – 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 the rising edge detection interrupts while writing a **0** to a bit position selects falling edge detection interrupts.

ENAB0 - ENAB2 – 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 **0**, the bit's edge detection interrupt is disabled. Note that this 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.

WS16C48 Programming Reference

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 WS16C48 or can serve as the basis from which application specific code can be derived.

Function Definitions

This section describes each of the functions contained in the driver. Where necessary, short examples will be provided to illustrate usage. Any application making use of any of the driver functions should include the header file UIS48.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(unsigned io_address);

Description

This function takes a single argument:

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 Pin

Syntax

int read_bit(int bit_number);

Description

This function takes a single argument:

bit_number – a value from 1 to 48 specifying 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 bit_number, int value);

Description

This function takes two arguments:

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 Pin

Syntax

void set_bit(int bit_number);

Description

This function takes a single argument:

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 Pin

Syntax

void clr_bit(int bit_number);

Description

This function takes a single argument:

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 bit_number, int polarity);

Description

This function takes two arguments:

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 a single argument:

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 bit_number);

Description

This function takes a single argument:

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

void get_int(void);

Description

This function requires no arguments.

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 function 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 be used in an application's ISE or can be used in the foreground to poll for bit transitions.

Sample Programs

There are three sample programs in source code form included on the PCM-UIO48B diskette. These programs are not useful by themselves but are provided to illustrate the usage of the I/O functions provided in UIO48.C.

FLASH.C

This program was compiled with Borland C/C++ version 3.1 on the command line with:
bcc flash.c uio48.c

This program illustrates the most basic usage of the WS16C48. It uses three functions from the driver code. The io_init() function is used to initialize the I/O functions and the set_bit() and clr_bit() functions are used to sequence through all 48 bits turning each on and then off in turn.

POLL.C

This program was compiled with Borland C/C++ version 3.1 on the command line with:
bcc poll.c uio48.c

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

INT.C

This program was compiled with Borland C/C++ version 3.1 on the command line with:
bcc int.c uio48.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 completed in the background during the interrupt service routine.

Summary

Links to the source code for all three programs as well as the I/O routines can be found in the Software Drivers & Examples Section of this manual. It is also provided in the C Source Code Listings Section of this manual. These I/O routines along with the sample program should provide for a good basis on which to build an application using the features of the WS16C48.

C Source Code Listings

```
/* UIO48.H

        Copyright 1996 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   : uio48.h
*
*       Project : PCM-UIO48 Software Samples/Examples
*
*       Date    : October 30, 1996
*
*       Revision: 1.00
*
*       Author  : Steve Mottin
*
*************************************************************************
*
*       Changes :
*
*       Date            Revision        Description
*       ________        ________        ___________________________________
*       10/30/96        1.00            Created
*
*************************************************************************
*/

#define RISING 1
#define FALLING 0

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

```
/* UIO48.C

        Copyright 1996 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   : uio48.c
*
*       Project : PCM-UIO48 Software Samples/Examples
*
*       Date    : October 30, 1996
*
*       Revision: 1.00
*
*       Author  : Steve Mottin
*
*************************************************************************
*
*       Changes :
*
*       Date            Revision        Description
*       ________        ________        ___________________________________
*       10/30/96        1.00            Created
*
*************************************************************************
*/

#include <dos.h>

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

unsigned base_port;

/* 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[6];

/*=======================================================================
===========
*                       INIT_IO
*
*  This function take a single argument :
*
*
*  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(unsigned io_address)
{
int x;

        /* Save the specified address for later use */

        base_port = io_address;

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

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

        /* Clear our image values as well */

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

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

        outportb(base_port+7,0x80);

        /* Clear all interrupt enables */

        outportb(base_port+8,0);
        outportb(base_port+9,0);
        outportb(base_port+0x0a,0);

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

}

/*=============================================================================
*
*                       READ_BIT
*
*
*  This function takes a single argument :
*
*
*  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 bit_number)
{
unsigned port;
int val;
```

```
        /* 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;

        /* 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 two arguments :
*
*
*  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 bit_number, int val)
{
unsigned port;
unsigned temp;
unsigned mask;


        /* 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;

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

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

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

        outportb(port,temp);
}

/*=================================================================
===========
*                       SET_BIT
*
*
*  This function takes a single argument :
*
*  bit_number :  The bit number to set.
*
*  This function sets the specified bit.
*
*=================================================================
==========*/

void set_bit(int bit_number)
{

        write_bit(bit_number,1);
}

/*=================================================================
===========
*                       CLR_BIT
*
*
*  This function takes a single argument :
*
*  bit_number :  The bit number to clear.
*
*  This function clears the specified bit.
*
*=================================================================
==========*/

void clr_bit(int bit_number)
{
        write_bit(bit_number,0);
}

/*=================================================================
===========
*
*                     ENAB_INT
*
```

```
*  This function takes two arguments :
*
*  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 bit_number, int polarity)
{
unsigned port;
unsigned temp;
unsigned mask;

        /* Adjust for 0 based numbering */

        --bit_number;

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

        port = (bit_number / 8) + base_port + 8;

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

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

        /* Turn on page 2 access */

        outportb(base_port+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+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+7,0x0);

}

/*================================================================
============
*
*                       DISAB_INT
*
* This function takes a single argument :
*
* 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 bit_number)
{
unsigned port;
unsigned temp;
unsigned mask;

        /* 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 + 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+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+7,0x0);
```

```
}

/*=============================================================================
==========
*
*                       CLR_INT
*
*  This function takes a single argument :
*
*  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 bit_number)
{
unsigned port;
unsigned temp;
unsigned mask;


        /* Adjust for 0 based numbering */

        --bit_number;

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

        port = (bit_number / 8) + base_port + 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+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+7,0x0);

}


/*=============================================================================
==========
*
*                       GET_INT
*
*  This function take no arguments.
*
*  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(void)
{
int temp;
int x;

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

        temp = inportb(base_port+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+7,0xc0);

        /* Read interrupt ID register for port 0 */

        temp = inportb(base_port+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+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+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+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+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+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+7,0);
        return 0;
}
```

```
/* FLASH.C

        Copyright 1996-2001 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 "uio48.h"

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

#define BASE_PORT 0x120


/* This is an utlra-simple demonstration program of some of the functions
   available in the UIO48 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(BASE_PORT);

        /* 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(x);
                        /* The wait time is subjective. We liked 100mS */
                        delay(100);
                        /* Now turn off the LED */
                        clr_bit(x);
                }
        }
        getch();
}
```

```
/* POLL.C

      Copyright 1996-2001 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 "uio48.h"


#define BASE_PORT 0x120

/* This program uses the edge detection interrupt capability of the
   WS16C48 to count transitions on the first 24 lines. It does this
   however, no 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[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(BASE_PORT);


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

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

```
        }

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

        for(x=1; x<25; x++)
        {
                gotoxy(1,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[x]);
                }
        }
        getch();
}

void check_ints()
{
int current;

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

        current = get_int();

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

        if(current == 0)
                return;

        /* Clear and rearm this one so we can get it again */

        clr_int(current);

        /* Tally a transition for this bit */

        ++int_counts[current];
}
```

```
/* INTS.C

        Copyright 1996-2001 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 "uio48.h"

#define BASE_PORT 0x120

/* 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[25];

/* Function declarations for local functions */

void check_ints(void);
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(BASE_PORT);

        /* 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(0x72);    /* Hardwired for IRQ10 */
```

```
/* Install out new interrupt handler */

   setvect(0x72,int_handler);

      /* Clear the transition count values and enable the falling edge
         interrupts.
      */

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


      /* Unmask the interrupt controller */

   outportb(0xa1,(inportb(0xa1) & 0xfb));  /* Unmask IRQ 10 */

      /* 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);
      }


      /* 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[x]);
               }
      }

      getch();

      /* Disable interrupts while we restore things */

      disable();

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

   outportb(0xa1,inportb(0xa1) | 0x02);    /* Mask IRQ 10 */

      /* Restore the old handler */
```

```
        setvect(0x72,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 current;

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

            current = get_int();

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

            while(current)
            {

                    /* Clear the current one so that it's ready for the next edge */

                    clr_int(current);

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

                    ++int_counts[current];

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

                    current = get_int();
            }

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

        outportb(0xa0,0x20);    /* Do non-specific EOI */
        outportb(0x20,0x20);
}
```

Cables

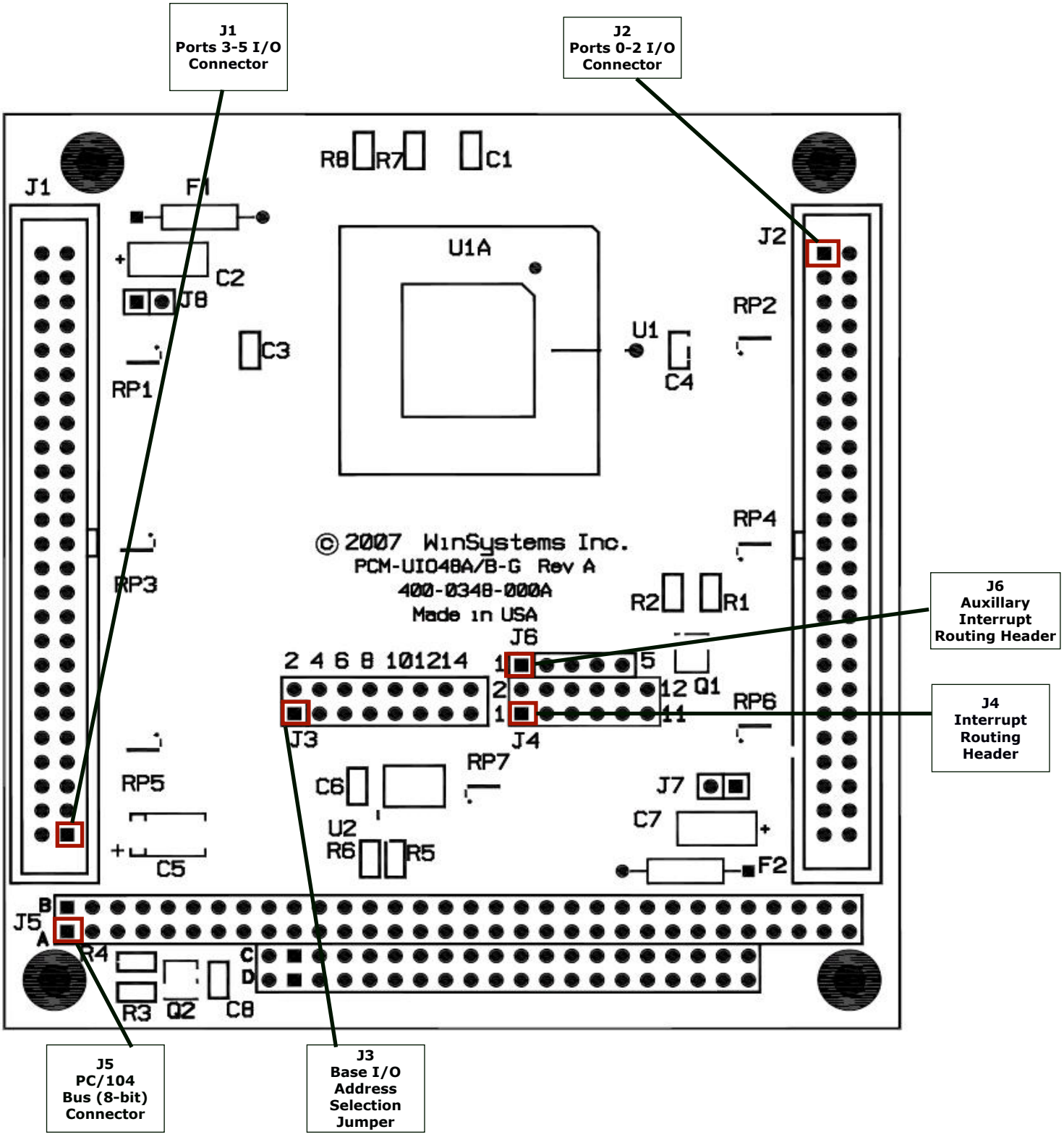
Part Number	Description
CBL-115-4	4-ft., Opto rack interface ***Only available for models:** PCM-UIO48B-16
CBL-129-4	4 ft., ribbon cable, 50 pin. Both ends with 50-pin socket termination ***Only available for models:** PCM-UIO48B-16

Software Drivers & Examples

Examples	
(For WS16C48 Digital I/O Chip)	
Linux Drivers - Kernel 2.2, 2.4	linux_uio48_96.zip
Linux Drivers - Kernel 2.6	uio48io_kernel_2.6.zip
DOS Example C Functions	uio48a.zip
Windows XP Driver	wsuio48_96xp.zip

Jumper Reference

Drawings ONLY - for more detailed information on these parts, refer to the descriptions shown previously in this manual.



NOTE: The reference line to each component part has been drawn to Pin 1, where applicable. Pin 1 is also highlighted with a red square, where applicable.

Digital I/O

J2

P2-7	**1 o o 2**	GND
P2-6	**3 o o 4**	GND
PS-5	**5 o o 6**	GND
PS-4	**7 o o 8**	GND
P2-3	**9 o o 10**	GND
P2-2	**11 o o 12**	GND
P2-1	**13 o o 14**	GND
P2-0	**15 o o 16**	GND
P1-7	**17 o o 18**	GND
P1-6	**19 o o 20**	GND
P1-5	**21 o o 22**	GND
P1-4	**23 o o 24**	GND
P1-3	**25 o o 26**	GND
P1-2	**27 o o 28**	GND
P1-1	**29 o o 30**	GND
P1-0	**31 o o 32**	GND
P0-7	**33 o o 34**	GND
P0-6	**35 o o 36**	GND
P0-5	**37 o o 38**	GND
P0-4	**39 o o 40**	GND
P0-3	**41 o o 42**	GND
P0-2	**43 o o 44**	GND
P0-1	**45 o o 46**	GND
P0-0	**47 o o 48**	GND
+5V	**49 o o 50**	GND

J1

P5-7	**1 o o 2**	SW1
P5-6	**3 o o 4**	SW3
P5-5	**5 o o 6**	GND
P5-4	**7 o o 8**	GND
P5-3	**9 o o 10**	GND
P5-2	**11 o o 12**	GND
P5-1	**13 o o 14**	GND
P5-0	**15 o o 16**	GND
P4-7	**17 o o 18**	GND
P4-6	**19 o o 20**	GND
P4-5	**21 o o 22**	GND
P4-4	**23 o o 24**	GND
P4-3	**25 o o 26**	GND
P4-2	**27 o o 28**	GND
P4-1	**29 o o 30**	GND
P4-0	**31 o o 32**	GND
P3-7	**33 o o 34**	GND
P3-6	**35 o o 36**	GND
P3-5	**37 o o 38**	GND
P3-4	**39 o o 40**	GND
P3-3	**41 o o 42**	GND
P3-2	**43 o o 44**	GND
P3-1	**45 o o 46**	-12V
P3-0	**47 o o 48**	+12V
+5V	**49 o o 50**	SWVCC

Interrupt Routing

J4

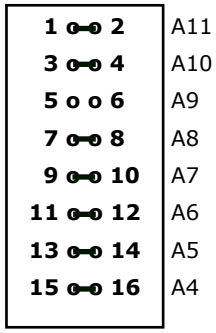
IRQ7	**1 o o 2**	
IRQ6	**3 o o 4**	
IRQ5	**5 o–o 6**	
IRQ4	**7 o o 8**	
IRQ3	**9 o o 10**	
IRQ2	**11 o o 12**	

J6

1 o	IRQ15
2 o	IRQ14
3 o	IRQ12
4 o	IRQ11
5 o	IRQ10

I/O Address Selection

I/O Base Address Select jumper **J3** shown jumpered for 200H



Specifications

Electrical

Bus Interface	:PC/104 (8-bit, optional 16-Bit extension)
VCC	:+5V ±5% @12 mA typical with no I/O connections
I/O Addressing	:12-Bit user jumperable base address. Each board uses 16 consecutive I/O addresses.

Mechanical

Dimensions	:3.6” X 3.8” (90 mm x 96 mm)
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 0.10” centers
Connectors	:50-pin 0.10” grid RN type IDH-50-LP

Environmental

Operating Temperature	:-40°C to +85° C
Noncondensing humidity	:5% to 95%

WARRANTY REPAIR INFORMATION

WARRANTY

WinSystems warrants to Customer 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. Notwithstanding anything herein to the contrary, this warranty granted by WinSystems to the Customer shall be for the sole benefit of the Customer, and may not be assigned, transferred or conveyed to any third party. 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 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 of 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, INCIDENTIAL 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

1. To obtain service under this warranty, obtain a return authorization number. In the United States, contact the WinSystems' Service Center for a return authorization number. Outside the United States, contact your local sales agent for a return authorization number.

2. You must send the product postage prepaid and insured. You must enclose the products in an anti-static bag to protect from damage by static electricity. WinSystems is not responsible for damage to the product due to static electricity.