PCM-UIO48/96 Digital I/O
Windows Device Driver Package

1 Introduction

1.1 The WinSystems PCM-UIO48 and PCM-UIO96 devices provide PC/104-compliant digital I/O peripheral modules. The digital I/O solutions utilize one or two WS16C48 Universal I/O Controller ASICs. Modules can be stacked to increase the I/O count up to a maximum of 192 (4 x 48-bit devices).

1.2 The WS16C48 ASIC supports 48 digital I/O lines addressed through six contiguous registers. Each I/O line is individually programmable for input, output, or output with read back operation. The ASIC supports up to 24 event sense lines which can sense a positive or negative transition on the input. These can be used to generate a system interrupt request.

1.3 The PCM-UIO48 driver package is designed for and has been verified with 32-bit and 64-bit versions of Microsoft Windows 7 and 10.

1.4 The PCM-UIO48 driver supports each WS16C48 ASIC used in the system. The driver must be loaded and configured for each instance of the ASIC.

2 Installation

2.1 Before installing any digital I/O device, verify that the board jumpers are configured for the desired I/O base address and interrupt. In the BIOS set-up menus, verify that the selected resources are not used by other devices.

2.2 The driver, support files, and console applications are supplied in a zip file. The following files are included:

a. pcmuio48.sys – Windows device driver
b. pcmuio48.inf – Windows installation file
c. pcmuio48.cat – Windows catalog file
d. WdfCoinstaller01011.dll – Windows co-installer
e. pcmuio48DLL.dll – Windows DLL
f. pcmuio48DLL.lib – Windows library file
g. pcmuio48DLL.h – driver include file
h. flash.cpp – Windows application source
i. poll.cpp – Windows application source
j. flash.exe – Windows application
k. poll.exe – Windows application

l. vcredist_x86 or vcredist_x64 - Microsoft Visual C++ Redistributable

2.3 Installation is accomplished via the 'Add legacy hardware' selection found in the Action menu of the Windows Device Manager. Navigate to the drive and folder containing the driver files and select *pcmuio48.inf*. The Windows installer will copy the *pcmuio48.sys* driver file to the appropriate directory in the Windows installation.

2.4 Installation is required for each 48-bit device installed. The PCM-UIO96 includes two separate devices so there must be two instances of the driver loaded. The I/O range in this case in 20h sequential bytes. Separate interrupts can be used or the same interrupt can be shared according to the jumper configuration of the board.

2.5 If multiple boards are stacked, the driver must be loaded for each 48-bit device. Each instance of the driver should be configured to match the jumper configuration.

2.6 In Device Manager, the PCM-UIO48 Device(s) will appear under the System Devices item. The desired hardware configuration can be selected under the Resources tab of the PCM-UIO48 Device Properties window. A reboot may be required after resource selection is complete.

2.7 The included console applications, *flash.exe* and *poll.exe,* can be used to verify driver installation and functionality. Usage of these programs is described later in this document.

3 Driver Overview and Architecture

3.1 The file *pcmuio48.sys* is a Windows Driver Foundation kernel-mode (KMDF) driver which facilitates access to the underlying hardware.

3.2 The file *pcmuio48DLL.dll* is a Windows Dynamic-Link Library which provides more user-friendly functions to access the device.

3.3 The driver utilizes the I/O Control (IOCTL) Request framework to control the register set of each WS16C48 ASIC. Data is passed to and from the driver utilizing input and output buffers.

4 Driver Usage

4.1 The *pcmuio48DLL.h* file included in the driver distribution contains the function definitions to be used by an application to communicate with the pcmuio48 driver. This file is included in Appendix A: pcmuio48DLL.h.

4.1 An application calls the function *InitializeSession* to open the driver. This is required before any of the other functions can be called. The following example opens the WinSystems *pcmuio48* driver. If a zero is returned, then the driver has been successfully initialized. Any other returned value indicates that an error has occurred and the device is unusable.

```
#define DEVICE 1  // access device 1

if (InitializeSession(DEVICE))
       printf("Error opening device.\n");
```

4.3 Once the driver is initialized, the other functions can be used to control the PCM-UIO48. Following is a description and sample code for each function. Each function requires a *device* parameter which selects the desired ASIC to access. The driver will support up to four devices.

For all functions, if a zero is returned, the function was successful. Otherwise there was an error and the function did not complete. Specific error codes are defined later in this document.

4.3.1 int ResetDevice(unsigned int device)

This function resets the device to a known state. All bits are defined as outputs at state zero and all interrupts are disabled. Any locked port is unlocked.

```
if (ResetDevice(2)) // reset device 2
       printf("Error resetting device.\n");
```

4.3.2 int SetIoMask(unsigned int device, unsigned int *portState)

Configures the mask for all ports provided in the memory location provided by the input array parameter *portState*. A mask bit defined as a zero is an input, and a mask bit defined as a one is an output.

```
unsigned int device = 1;
unsigned int mask[6]; // 48 I/O = 6 Ports

mask[0] = 0xFF; // all bits output
mask[1] = 0x00; // all bits input
mask[2] = 0xF0; // upper nibble output, lower nibble input
mask[3] = 0xFF; // all bits output
mask[4] = 0xAA; // alternating input/output bits
mask[5] = 0x55; // alternating input/output bits

if (SetIoMask(device, mask))
       printf("Error configuring port masks.\n");
```

4.3.3 int GetIoMask(unsigned int device, unsigned int *portState)

Retrieves the mask for all ports and stores them in the memory locations provided by the array parameter *portState*. A mask bit defined as a zero is an input, and a mask bit defined as a one is an output.

```
unsigned int device = 3;
unsigned int mask[6]; // 48 I/O = 6 Ports

if (GetIoMask(device, mask))
       printf("Error reading port masks.\n");
```

4.3.4 int ReadAllPorts(unsigned int device, unsigned int *readValueArray)

Reads the current value of all available ports and stores them in the memory locations provided by the array parameter *readValueArray*.

```
unsigned int device = 2;
unsigned int read_value[6]; // 48 I/O = 6 Ports

if (ReadAllPorts(device, read_value))
       printf("Error reading all ports.\n");
```

4.3.5 int ReadPort(unsigned int device, int port, unsigned int *readValue)

Reads the current value of the selected port and stores it in the memory location provided by the parameter *readValue*.

```
unsigned int device = 1;
unsigned int read_value;
int port = 1;

if (ReadPort(device, port, &read_value))
       printf("Error reading device %d port %d.\n", device, port);
else
       printf("Device %d Port %d = 0x%0x\n", device, port, read_value);
```

4.3.6 int ReadBit(unsigned int device, int bit, unsigned int *bitValue)

Reads the current value (0 or 1) of the selected bit and stores it in the memory location provided by the parameter *bitValue*.

```
unsigned int device = 1;
unsigned int bit_value;
int bit = 40;

if (ReadBit(device, bit, &bit_value))
       printf("Error reading device %d bit %d.\n", device, bit);
else
       printf("Device %d Bit %d = %d\n", device, port, bit_value);
```

4.3.7 int SetBit(unsigned int device, int bit)

Sets the selected bit. If the bit is part of a locked port, an ACCESS_ERROR value is returned.

```
unsigned int device = 4;
int bit = 24;

if (SetBit(device, bit))
       printf("Error setting device %d bit %d.\n", device, bit);
```

4.3.8 int ClearBit(unsigned int device, int bit)

Clears the selected bit. If the bit is part of a locked port, an ACCESS_ERROR value is returned.

```
unsigned int device = 1;
int bit = 0;
```

```
if (ClearBit(device, bit))
	printf("Error clearing device %d bit %d.\n", device, bit);
```

4.3.9 int WritePort(unsigned int device, int port, unsigned int writeValue)

Writes the value in the parameter *writeValue* to the selected port. If the port is locked, an ACCESS_ERROR value is returned.

```
unsigned int device = 2;
unsigned int write_value = 0x55;
int port = 4;

if (WritePort(device, port, write_value))
	printf("Error writing to device %d port %d.\n", device, port);
```

4.3.10 int WriteBit(unsigned int device, int bit, unsigned int bitValue)

Writes the value specified by the parameter *bitValue* (0 or 1) to the selected bit. If the bit is part of a locked port, an ACCESS_ERROR value is returned.

```
unsigned int device = 1;
int bit = 32;
unsigned int bit_value = 1;

if (WriteBit(device, bit, bit_value))
	printf("Error writing device %d bit %d.\n", device, bit);
```

4.3.11 int EnableInterrupt(unsigned int device, int bit, int edge)

This function enables interrupts for the selected bit. The edge parameter selects a rising or falling edge trigger for the interrupt. An enum value is provided which defines valid values for the parameter *edge* (FALLING_EDGE = 0 and RISING_EDGE = 1).

```
unsigned int device = 2;
int bit = 0;

if (EnableInterrupt(device, bit, RISING_EDGE))
	printf("Error enabling interrupts for device %d bit %d.\n", device,
bit);
else
	printf("Interrupts enabled for device %d bit %d.\n", device, bit);
```

4.3.12 int DisableInterrupt(unsigned int device, int bit)

This function disables interrupts for the selected bit.

```
unsigned int device = 2;
int bit = 23;

if (DisableInterrupt(bit))
	printf("Error disabling interrupts for device %d bit %d.\n", device,
bit);
else
```

```
printf("Interrupts disabled for device %d bit %d.\n", device, bit);
```

4.3.13 int GetInterrupt(unsigned int device, unsigned int *irqArray)

This function retrieves the interrupt status for all bits and stores them in the memory locations provided by the parameter *irqArray*. Any bit that is set indicates that an interrupt has occurred on that bit. After being read, the interrupt status on all interruptible bits is reset to zero.

```
unsigned int device = 4;
unsigned int irq[3]; // 24 IRQ = 3 Ports

if (GetInterrupt(device, irq))
	printf("Error retrieving interrupts for device %d.\n", device);
```

4.3.14 int WaitForInterrupt(unsigned int device, unsigned int *irqArray, unsigned long timeout)

This function forces the driver to wait for an interrupt on any bit that has been enabled for interrupts. If an interrupt already exists on a bit, the function will act like the GetInterrupt and immediately return and store the interrupt status in the memory locations provided by the parameter *irqArray*.

If no interrupts are present, the function will wait until an interrupt does occur. This function will not stop the execution of driver functions in another thread.

The *timeout* parameter provides a safeguard against system lockup. The value entered provides a time out in milliseconds. The pending operation is canceled. To disable the timeout feature, use INFINITE as the timeout parameter.

If the session is terminated before an interrupt occurs, the IO request will be automatically terminated.

```
unsigned int device = 1;
unsigned int irq[3];  // 24 IRQ = 3 Ports
long timeout = 0x1000;  // timeout = 4096 ms

if (WaitForInterrupt(device, irq, timeout))
	printf("Error waiting for interrupts from device %d.\n", device);
```

4.3.15 int LockPort(unsigned int device, int port)

Locks the selected port which prevents writing to any bits in that port. The register can still be read.

```
unsigned int device = 1;
int port = 5;

if (LockPort(device, port))
	printf("Error locking device %d port %d.\n", device, port);
else
	printf("Device %d Port %d locked for writing.\n", device, port);
```

4.3.16 int UnlockPort(unsigned int device, int port)

Unlocks the selected port which enables writing to any bits in that port.

```
int port = 0;

if (UnlockPort(device, port))
       printf("Error unlocking device %d port %d.\n", device, port);
else
       printf("Device %d Port %d unlocked.\n", device, port);
```

4.3.17 int CloseSession(unsigned int device)

This function is used to disable the pcmuio48 device and close the driver when complete. If a zero is returned, the timer is disabled and the driver is closed. Otherwise there was an error and the driver is still open.

```
unsigned int device = 1;

if (CloseSession(device))
       printf("Error closing driver.\n");
```

4.4 Every function returns a zero or a positive integer value indicating success or failure. If a zero is returned, the function has completed successfully. If a failure occurs, the specific value returned provides more clarity as to the failure mechanism.

4.4.1 DRIVER_ERROR (1)

This error indicates that some function within the driver has failed. This error indicates that one of the IOCTL calls within the driver itself has not completed successfully. Using Windows Device Manager, verify that the driver is loaded and has no resource conflicts.

4.4.2 ACCESS_ERROR (2)

This error indicates the following conditions are present:

- The driver has tried to write to a bit defined as an input
- The driver attempts to enable interrupts for a bit defined as an output
- The driver attempts to write to a locked port
- The driver attempts to change a bit in a locked port
- The driver is unable to start a WaitForInterrupt call

Verify that the port mask has been set as desired or unlock the port before attempting to change its state.

4.4.3 INVALID_HANDLE (3)

This error indicates that the driver has not initialized or closed. The driver attempts to obtain a handle to the PCM-UIO48 device, and this error indicates that the handle was not obtained. Verify that the driver has loaded successfully.

4.4.4 INVALID_PARAMETER (4)

This error indicates that one of the parameters in a DLL function is out of bounds.

4.4.5 TIMEOUT_ERROR (5)

This error indicates that the WaitForInterrupt function has exceeded the provided timeout value before an interrupt occurred.

5 Sample Applications

There are two sample Windows console applications provided in the driver package, *flash* and *poll*. The source code for the applications are provided in Appendix B: flash.cpp and Appendix C: poll.cpp.

5.1 Flash

The *flash* sample application is a console application that sets and clears each bit in sequential order. The defined value MAX_DEV allows multiple devices to be accessed. Each device is opened, all bits are configured as outputs, each bit is toggled, and then each device is closed. If any error occurs during execution, the error is reported and the application is terminated.

5.2 Poll

The *poll* sample application is a console application that tests interrupts for a single device. The device is opened and all bits of the first three ports are configured as inputs. Interrupts are enabled for each input bit alternating between rising and falling edge. A separate thread is created that calls the WaitForInterrupt function. If any of the input bits are toggled with the proper edge polarity, the specific interrupt is displayed on the console. Any keystroke will terminate the program and report the total number of interrupts.

Appendix A: pcmuio48DLL.h

```
//****************************************************************************
//
//     Copyright 2017 by WinSystems Inc.
//
//****************************************************************************
//
//     Name    : pcmuio48DLL.h
//
//     Project : PCM-UIO48 Windows DLL
//
//     Author  : Paul DeMetrotion
//
//****************************************************************************
//
//       Date        Rev                    Description
//     --------    -------     ------------------------------------------
//     10/06/17      4.0           Original Release of DLL
//
//****************************************************************************

#ifndef _PCMUIO48_DLL_H_
#define _PCMUIO48_DLL_H_

#if defined DLL_EXPORT
#define DECLDIR __declspec(dllexport)
#else
#define DECLDIR __declspec(dllimport)
#endif

extern "C"
{
    DECLDIR int InitializeSession(unsigned int device);
    DECLDIR int ResetDevice(unsigned int device);
    DECLDIR int SetIoMask(unsigned int device, unsigned int *portState);
    DECLDIR int GetIoMask(unsigned int device, unsigned int *portState);
    DECLDIR int ReadAllPorts(unsigned int device, unsigned int *readValueArray);
    DECLDIR int ReadPort(unsigned int device, int port, unsigned int *readValue);
    DECLDIR int ReadBit(unsigned int device, int bit, unsigned int *bitValue);
    DECLDIR int SetBit(unsigned int device, int bit);
    DECLDIR int ClearBit(unsigned int device, int bit);
    DECLDIR int WritePort(unsigned int device, int port, unsigned int writeValue);
    DECLDIR int WriteBit(unsigned int device, int bit, unsigned int bitValue);
    DECLDIR int EnableInterrupt(unsigned int device, int bit, int edge);
    DECLDIR int DisableInterrupt(unsigned int device, int bit);
    DECLDIR int GetInterrupt(unsigned int device, unsigned int *irqArray);
    DECLDIR int WaitForInterrupt(unsigned int device, unsigned int *irqArray, unsigned
long timeout);
    DECLDIR int LockPort(unsigned int device, int port);
    DECLDIR int UnlockPort(unsigned int device, int port);
    DECLDIR int CloseSession(unsigned int device);
}

typedef enum {
    SUCCESS = 0,
    DRIVER_ERROR,
```

```
    ACCESS_ERROR,
    INVALID_HANDLE,
    INVALID_PARAMETER,
    TIMEOUT_ERROR
} ErrorCodes;

typedef enum {
    FALLING_EDGE = 0,
    RISING_EDGE
} IrqEdge;

#endif
```

Appendix B: flash.cpp

```
//****************************************************************************
//
//     Copyright 2017 by WinSystems Inc.
//
//****************************************************************************
//
//     Name    : flash.cpp
//
//     Project : PCM-UIO48/96 Console Application
//
//     Author  : Paul DeMetrotion
//
//****************************************************************************
//
//       Date        Rev                    Description
//     --------    -------     ------------------------------------------
//     10/06/17      1.0           Original Release
//
//****************************************************************************

#include <stdio.h>
#include <stdlib.h>
#include <tchar.h>
#include <windows.h>
#include "pcmuio48DLL.h"

#define MAJOR_VER   1
#define MINOR_VER   0
#define MAX_DEV     2

int _tmain(int argc, _TCHAR* argv[])
{
    unsigned int portState[6] = { 0xff, 0xff, 0xff, 0xff, 0xff, 0xff };  // define all as
outputs
    int dllReturn = SUCCESS;

    printf("PCM-UIO48 Application : flash\n");
    printf("Version %d.%d\n\n", MAJOR_VER, MINOR_VER);

    for (int dev = 1; dev <= MAX_DEV; dev++)
    {
        dllReturn = InitializeSession(dev);

        if (dllReturn)
        {
            printf("Error initializing device %d.\n", dev);
            exit(dllReturn);
        }
        else
        {
            printf("Device %d opened.\n", dev);
        }
    }

    for (int dev = 1; dev <= MAX_DEV; dev++)
```

```
    {
        dllReturn = SetIoMask(dev, portState);

        if (dllReturn)
        {
            printf("Error setting I/O mask for device %d.\n", dev);
            exit(dllReturn);
        }
    }

    for (int dev = 1; dev <= MAX_DEV; dev++)
    {
        printf("Flashing bits for device %d...\n", dev);

        for (int bit = 0; bit < 48; bit++)
        {
           dllReturn = SetBit(dev, bit);

           if (dllReturn)
           {
               printf("\nError setting bit %d on device %d.\n", bit, dev);
               break;
           }

           Sleep(200);

           if (ClearBit(dev, bit))
           {
               printf("\nError clearing bit %d on device %d.\n", bit, dev);
               break;
           }

           Sleep(200);
        }
    }

    for (int dev = 1; dev <= MAX_DEV; dev++)
    {
        dllReturn = CloseSession(dev);

        if (dllReturn)
        {
            printf("Error closing device %d.\n", dev);
            exit(dllReturn);
        }
        else
        {
            printf("Device %d closed.\n", dev);
        }
    }

    return(0);
}
```

Appendix C: poll.cpp

```
//****************************************************************************
//
//     Copyright 2017 by WinSystems Inc.
//
//****************************************************************************
//
//     Name    : poll.cpp
//
//     Project : PCM-UIO48/96 Console Application
//
//     Author  : Paul DeMetrotion
//
//****************************************************************************
//
//       Date        Rev                    Description
//     --------    -------     ------------------------------------------
//     10/06/17      1.0           Original Release
//
//****************************************************************************

#include <windows.h>
#include <conio.h>
#include <stdio.h>
#include <stdlib.h>
#include <tchar.h>
#include <strsafe.h>
#include "pcmuio48DLL.h"

DWORD WINAPI MyThreadFunction(LPVOID lpParam);

#define MAJOR_VER   1
#define MINOR_VER   0
#define DEVICE      1

volatile int irq_count = 0;
volatile int exit_flag = 0;
char line[80];

int _tmain(int argc, _TCHAR* argv[])
{
    HANDLE hThread;
    DWORD dwThreadId;
    int ch;
    unsigned int portState[6] = { 0x00, 0x00, 0x00, 0xff, 0xff, 0xff };  // port1 + port2
+ port3 inputs
    unsigned long timeout = 0x100; // 4096 msec
    int dllReturn = SUCCESS;

    printf("PCM-UIO48 Application : poll\n");
    printf("Version %d.%d\n\n", MAJOR_VER, MINOR_VER);

    dllReturn = InitializeSession(DEVICE);

    if (dllReturn)
    {
```

```
    printf("Error initializing session.\n");
    exit(dllReturn);
}
else
{
    printf("Device %d opened.\n", DEVICE);
}

dllReturn = SetIoMask(DEVICE, portState);

if (dllReturn)
{
    printf("Error setting I/O mask.\n");
    exit(dllReturn);
}

// enable rising edge interrupts for odd bits
// enable falling edge interrupts for even bits
for (int i = 0; i < 24; i++) {
    if (i % 2)
        dllReturn = EnableInterrupt(DEVICE, i, RISING_EDGE);
    else
        dllReturn = EnableInterrupt(DEVICE, i, FALLING_EDGE);

    if (dllReturn)
    {
        printf("Error enabling interrupts for bit %d.\n", i);
        exit(dllReturn);
    }
}

// Create the thread to begin execution on its own.
hThread = CreateThread(
    NULL,                   // default security attributes
    0,                      // use default stack size
    MyThreadFunction,       // thread function name
    &timeout,               // argument to thread function
    0,                      // use default creation flags
    &dwThreadId);           // returns the thread identifier

// if CreateThread fails, terminate execution
if (hThread == NULL)
{
    printf("Cannot create thread!\n");
    exit(1);
}

// print menu
printf("Toggle bits to generate interrupts\n");
printf("Odd bits configured for rising edge interrutps\n");
printf("Even bits configured for falling edge interrutps\n");
printf("Hit any key when done\n\n");

// wait for key stroke
while (!_kbhit()) ;
ch = _getch();

// done so set flag
```

```
    exit_flag = 1;

    // disable interrupts
    for (int i = 0; i < 24; i++)
    {
        dllReturn = DisableInterrupt(DEVICE, i);

        if (dllReturn)
        {
            printf("Error disabling interrupt on bit %d.\n", i);
            exit(dllReturn);
        }
    }

    // Wait thread terminates
    WaitForSingleObject(hThread, INFINITE);

    // close thread handle
    CloseHandle(hThread);

    // finish
    dllReturn = CloseSession(DEVICE);

    if (dllReturn)
    {
        printf("Unable to close device.\n");
        exit(dllReturn);
    }
    else
        printf("PCM-UIO48 device closed.\n");

    // display our event count total
    printf("\nInterrupt count = %d\r", irq_count);
}

DWORD WINAPI MyThreadFunction(LPVOID lpParam)
{
    BYTE irqBit = 0;
    unsigned int irqArray[3];
    int dllReturn = SUCCESS;
    int count = 0;
    unsigned long *timeout = (unsigned long*) lpParam;

    while (1)
    {
        // put THIS process to sleep until interrupt occurs
        dllReturn = WaitForInterrupt(DEVICE, irqArray, *timeout);

        // exit loop if program is terminating
        if (exit_flag)
            break;

        // restart loop if error
        if (dllReturn)
            continue;

        // determine interrupt bit
        if (irqArray[0]) {
```

```
            while (!(irqArray[0] & (1 << count)))
                count++;

            irqBit = count + 1;
        }
        else if (irqArray[1]) {
            while (!(irqArray[1] & (1 << count)))
                count++;

            irqBit = count + 9;
        }
        else if (irqArray[2]) {
            while (!(irqArray[2] & (1 << count)))
                count++;

            irqBit = count + 17;
        }

        if (irqBit)
            printf("Interrupt on bit %d\n", irqBit);
        else
            printf("No Interrupt\n");

        count = 0;
        irq_count++;
    }

    return 0;
}
```