ADADIO

16-Bit: 8 Ch Analog Input, 4 Ch Analog Output,
8-Bit Digital I/O

Windows 98\NT\2K\XP Driver User Manual

Manual Revision: July 8, 2003

General Standards Corporation
8302A Whitesburg Drive
Huntsville, AL 35802
Phone: (256) 880-8787
Fax: (256) 880-8788
URL: http://www.generalstandards.com
E-mail: sales@generalstandards.com
E-mail: support@generalstandards.com

Preface

Additional copies of this manual or other literature may be obtained from:

General Standards Corporation
8302A Whitesburg Dr.
Huntsville, Alabama 35802
Phone: (256) 880-8787
FAX: (256) 880-8788
URL: http://www.generalstandards.com
E-mail: sales@generalstandards.com

General Standards Corporation makes no warranty of any kind with regard to this material, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. Although extensive editing and reviews are performed before release to ECO control, **General Standards Corporation** assumes no responsibility for any errors that may exist in this document. No commitment is made to update or keep current the information contained in this document.

General Standards Corporation does not assume any liability arising out of the application or use of any product or circuit described herein, nor is any license conveyed under any patent rights or any rights of others.

General Standards Corporation assumes no responsibility for any consequences resulting from omissions or errors in this manual or from the use of information contained herein.

General Standards Corporation reserves the right to make any changes, without notice, to this product to improve reliability, performance, function, or design.

The Purchaser of this software may use or modify in source form the subject software, but not to re-market or distribute it to outside agencies or separate internal company divisions. The software, however, may be embedded in the Purchaser's distributed software. In the event the Purchaser's customers require the software source code, then they would have to purchase their own copy of the software.

General Standards Corporation makes no warranty of any kind with regard to this software, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose and makes this software available solely on an "as-is" basis. **General Standards Corporation** reserves the right to make changes in this software without reservation and without notification to its users.

GSC is a trademark of **General Standards Corporation**.

PLX and PLX Technology are trademarks of PLX Technology, Inc.

Table of Contents

1. Scope

The Purpose of this document is to describe how to interface with the ADADIO Windows Driver API developed by General Standards Corporation (GSC). This software provides the interface between the “Application Software” and the ADADIO board.

The ADADIO Driver API Software executes under control of the Windows Operating System. The ADADIO is implemented as a standard Windows driver API written in “C” programming language. The ADADIO Driver API Software is designed to operate on CPU boards containing x86 processors.

The ADADIO Driver consists of a Windows driver with an interface layer (GSC API) to simplify the interface to the PLX Driver. While an application may interface directly to the PLX driver, interfacing to the GSC API layer, will simplify the application software development.

2. Hardware Overview

The PMC-ADADIO board is a single width PCI mezzanine card (PMC) that provides system analog input/output capability for the PCI bus. In addition to containing eight analog input channels and four analog output channels, the board also has a general-purpose byte-wide digital port. The board is functionally compatible with the IEEE PCI local bus specification Revision 2.2, is mechanically compatible with the IEEE compact mezzanine card (CMC) specification, and supports the "plug-nplay" initialization concept.

Selftest networks permit all channels to be calibrated automatically to a single internal voltage reference. Offset and gain trimming of the output channels are performed by calibration DAC's that are loaded with channel correction values during initialization. The correction values are determined during auto calibration, and are stored in nonvolatile EEprom for subsequent transfer to the calibration DAC's. Either auto calibration or initialization can be invoked at any time by asserting a single control bit in the board control register.

The board is designed for minimum off-line maintenance, and includes internal monitoring features that eliminate the need for disconnecting or removing the module from the system for calibration. All analog input and output system connections are made through a single 68-pin I/O connector. Power requirements consist of +5 VDC, in compliance with the PCI specification, and operation over the specified temperature range is achieved with conventional convection cooling.

3. Referenced Documents

The following documents provide reference material for the 16AO12 board:

- PMC-ADADIO User's Manual – GSC
- PLX Technology, Inc. PCI 9080 PCI Bus Master Interface Chip data sheet.

4. General Standards API

This section describes the interface to the ADADIO GSC API. The ADADIO GSC API isolates the user from operating system specific requirements, allowing the API to be used with all Windows operating systems (98\NT\W2K\XP).

The ADADIO Win Driver provides an interface to an ADADIO card and a Windows application, which run on a x86 target processor. The driver is installed and devices are created when the driver is started during boot up. The functions of the driver can then be used to access the board. Devices are created with the name “board x” where “x” is the device number. Device numbers start at 1 and for each board found the device number will increment.

Included in the board driver software is a menu driven board application program. This program is delivered undocumented and unsupported but may be used to exercise the card and the device driver. It can also be used as an example for programming the ADADIO device.

The user interfaces to the GSC API at the basic level with the following functions:

- Find Boards() - Detects all PLX Devices connected via the PCI Bus.
- Get Handle() - Opens a driver interface to one ADADIO card.
- Readlocal32() - Reads local registers from one ADADIO card.
- Writelocal32() - Writes to local Registers of one ADADIO card.
- Close Handle() - Closes a driver interface to one ADADIO card.

The user MUST call Find Boards to determine what PLX devices are installed in the system, and get the associated board number. The user then calls the Get Handle function with each board number to be used. This function obtains a handle to the device and initializes the device parameters within the API / driver. The user is then free (assuming no errors) to write / read the registers as desired. The user should always call Close Handle when done to free resources prior to exiting.

The function definitions and parameters are defined in the following paragraphs of this section.

4.1 ADADIO_FindBoards()

Detects all PLX Devices connected via the PCI Bus.

Prototype:

```
U32  ADADIO_FindBoards      (char       *pDeviceInfo,
                             U32        *ulError);
```

Returns – Total number of PLX boards found in the system or –1L if error or no boards found.

Where:

pDeviceInfo – Contains “Board #: Bus: Slot: Type: Ser#” info for PLX boards found.

ulError – Returns 0 or error code. Refer to tools.h for a list of error codes.

4.2 ADADIO_Get_Handle

Initializes Handle for the passed board number IN THE DRIVER.

Prototype:

```
U32  ADADIO_Get_Handle      (U32        *ulError,
                             U32        BoardNumber);
```

Returns – Error code if invalid board number passed (0, >31), else # boards.

Where:

BoardNumber – Defines board number to be used by the driver for a particular device.

ulError – Returns 0 or error code. Refer to tools.h for a list of error codes.

4.3 ADADIO_Read_Local32

Read a value from the board local register.

Prototype:

```
U32  ADADIO_Read_Local32   (U32     BoardNumber,
                            U32     *ulError,
                            U16     iRegister);
```

Returns – Value read from the register.

Where:

BoardNumber – Defines board number to be used by the driver for a particular device.

ulError – Returns 0 or error code. Refer to tools.h for a list of error codes.

iRegister – Register to read. Values defined in ADADIOintface.h

```
BCR             0x00
DIO             0x04
A_OUT0          0x08
A_OUT1          0x0C
A_OUT2          0x10
A_OUT3          0x14
A_IN_BUFF       0x18
SAMP_RATE       0x1C
```

4.4 ADADIO_Write_Local32

Write a value to the board local register.

Prototype:

```
void  ADADIO_Write_Local32  (U32    BoardNumber,
                            U32    *ulError,
                            U16    iRegister
                            U32    uiValue);
```

Returns – N/A

Where:

BoardNumber – Defines board number to be used by the driver for a particular device.

ulError – Returns 0 or error code. Refer to tools.h for a list of error codes.

iRegister – Register to write. Values defined in ADADIOintface.h

BCR	0x00
DIO	0x04
A_OUT0	0x08
A_OUT1	0x0C
A_OUT2	0x10
A_OUT3	0x14
A_IN_BUFF	0x18
SAMP_RATE	0x1C

uiValue – Value to write to the selected register.
Refer to the ADADIO user manual for all register / bit definitions.

4.5 ADADIO_Close_Handle

Closes the device handle and frees the resources.

Prototype:

```
void  ADADIO_Close_Handle   (U32     BoardNumber,
                             U32     *ulError);
```

Returns – N/A

Where:

BoardNumber – Defines board number to be used by the driver for a particular device.

ulError – Returns 0 or error code. Refer to tools.h for a list of error codes.

4.6 Interface Functions

These functions allow the user to perform certain operations on the board, without having to keep track of individual register values and bit definitions.

4.6.1 ADADIO_Initialize

Perform a reset on the board. All register values are set to defaults. This Function does NOT wait for Initialization to complete, such that multiple boards can be initialized without waiting for each to finish.

Prototype:

```
void  ADADIO_Initialize  (U32    BoardNumber,
                          U32    *ulError);
```

Returns – N/A

Where:

BoardNumber – Defines board number to be used by the driver for a particular device.

ulError – Returns 0 or error code. Refer to tools.h for a list of error codes.

4.6.2 ADADIO_Autocal

Perform an auto calibration on the board. This operation generates new calibration correction values which are stored in nonvolatile EEprom. This Function does NOT wait for Autocal to complete, such that multiple boards can be calibrated without waiting for each to finish.

Prototype:

```
void ADADIO_Autocal   (U32      BoardNumber,
                       U32      *ulError);
```

Returns – N/A

Where:

BoardNumber – Defines board number to be used by the driver for a particular device.

ulError – Returns 0 or error code. Refer to tools.h for a list of error codes.

4.6.3 ADADIO_Set_Input_Mode

Sets the input mode of the board: Differential, Single-Ended, Selftest (zero or Vref) or output channel (0-3) monitor.

Prototype:

```
void  ADADIO_Set_Input_Mode(U32     BoardNumber,
                            U32     *ulError
                            U32     ulInputMode);
```

Returns – N/A

Where:

BoardNumber – Defines board number to be used by the driver for a particular device.

ulError – Returns 0 or error code. Refer to tools.h for a list of error codes.

ulInputMode – Valid values: 0 – 7
- 0 = Single-Ended Continuous
- 1 = Single-Ended Burst
- 2 = Differential Continuous
- 3 = Differential burst
- 4 = LoopBack Selftest
- 5 = +Vref Selftest
- 6 = Reserved
- 7 = Zero Selftest

4.6.4 ADADIO_Set_Output_Mode

Sets the output mode of the board, immediate or strobed.

Prototype:

```
void  ADADIO_Set_Output_Mode    (U32    BoardNumber,
                                 U32    *ulError
                                 U32    ulOutputMode);
```

Returns – N/A

Where:

BoardNumber – Defines board number to be used by the driver for a particular device.

ulError – Returns 0 or error code. Refer to tools.h for a list of error codes.

ulOutputMode – Valid values: 0 = immediate, 1 = strobed.

4.6.5 ADADIO_Clear_Input_Buffer

Clears all data from the active input buffer.

Prototype:

```
void  ADADIO_Clear_Input_Buffer    (U32     BoardNumber,
                                    U32     *ulError);
```

Returns – N/A

Where:

BoardNumber – Defines board number to be used by the driver for a particular device.

ulError – Returns 0 or error code. Refer to tools.h for a list of error codes.

4.6.6 ADADIO_Enable_Outputs

Enables or disables the analog Outputs.

Prototype:

```
void  ADADIO_Enable_Outputs     (U32     BoardNumber,
                                 U32     *ulError
                                 U32     ulOutputEnable);
```

Returns – N/A

Where:

BoardNumber – Defines board number to be used by the driver for a particular device.

ulError – Returns 0 or error code. Refer to tools.h for a list of error codes.

ulOutputEnable – Valid values: 0 = disable, 1 = enable.

4.6.7 ADADIO_EnableInterrupt

Enables the desired interrupt in the local register, and for the PCI bus. See ADADIO User manual for interrupt sources.

Prototype:

```
U32  ADADIO_EnableInterrupt  (U32     BoardNumber,
                              U32     ulValue,
                              U32     *ulError);
```

Returns – Interrupt value set.

Where:

BoardNumber – Defines board number to be used by the driver for a particular device.

ulValue – The desired interrupt value to set, valid for 0 – 7. See Manual for definitions.

ulError – Returns 0 or error code. Refer to tools.h for a list of error codes.

4.6.8 ADADIO_DisableInterrupt

Disables interrupt in the local register, and for the PCI bus.

Prototype:

```
void  ADADIO_DisableInterrupt(U32     BoardNumber,
                              U32     ulValue,
                              U32     *ulError);
```

Returns – N/A.

Where:

BoardNumber – Defines board number to be used by the driver for a particular device.

ulValue – The desired interrupt value to clear, valid for 0 – 7. See Manual for definitions.

ulError – Returns 0 or error code. Refer to tools.h for a list of error codes.

4.6.9 ADADIO_CheckInterruptStatus

Querys the interrupt status for the device, and clears any flags which are set.

Prototype:

```
U32 ADADIO_CheckInterruptStatus   (U32     BoardNumber,
                                   U32     *ulError);
```

Returns – A (1) if interrupt occurred, 0 otherwise

Where:

BoardNumber – Defines board number to be used by the driver for a particular device.

ulError – Returns 0 or error code. Refer to tools.h for a list of error codes.

4.6.10 ADADIO_Open_DMA_Channel

Opens the desired DMA channel for transferring data from the board to user array.

Prototype:

```
void  ADADIO_Open_DMA_Channel  (U32      BoardNumber,
                                U32      ulChannel,
                                U32      *ulError);
```

Returns – N/A.

Where:

BoardNumber – Defines board number to be used by the driver for a particular device.

uiValue – The desired channel to open, currently valid for channel 0, 1.

ulError – Returns 0 or error code. Refer to tools.h for a list of error codes.

4.6.11 ADADIO_DMA_FROM_Buffer

Transfers the desired number of WORDS from the board input FIFO buffer.

Prototype:

```
U32  ADADIO_DMA_FROM_Buffer   (U32     BoardNumber,
                               U32     ulChannel,
                               U32     ulWords,
                               U32*    uData,
                               U32     *ulError);
```

Returns – WORDS transferred if no error.

Where:

BoardNumber – Defines board number to be used by the driver for a particular device.

ulChannel – The DMA channel previously opened, currently valid for channel 0 .

ulWords – Number of WORDS to transfer. (BYTES = ulWords*4).

ulError – Returns 0 or error code. Refer to tools.h for a list of error codes.

4.6.12 ADADIO_Close_DMA_Channel

Closes the desired DMA channel.

Prototype:

```
void  ADADIO_Close_DMA_Channel (U32     BoardNumber,
                                U32     ulChannel,
                                U32     *ulError);
```

Returns – N/A.

Where:

BoardNumber – Defines board number to be used by the driver for a particular device.

uiValue – The desired channel to close, currently valid for channel 0,1.

ulError – Returns 0 or error code. Refer to tools.h for a list of error codes.

4.6.13 ADADIO_Attach_Interrupt

Attaches a user supplied handle to an interrupt which can be used in WaitForSingleObject for notification when the interrupt occurs. A sample use is provided in the Autocal function of the example program.

The notification is a 'one-shot', i.e. single event. To repeatedly receive notification, reattach to the interrupt upon notification. i.e.

```
 ADADIO_Attach_Interrupt(ulBdNum, &myHandle, 0x01, &ulErr);
…
… Setup and code to cause interrupt to happen
…
loop begin
 EventStatus = WaitForSingleObject(myHandle,10 * 1000);

…
 switch(EventStatus)
 {
    case WAIT_OBJECT_0:
        … code to perform desired action
        ADADIO_Attach_Interrupt(ulBdNum, &myHandle, 0x01, &ulErr);
        break;
    default:
        cprintf("Interrupt was NOT requested...");
        break;
 }
loop end
```

Prototype:

```
void  ADADIO_Attach_Interrupt     (U32       BoardNumber,
                                   HANDLE    userHandle,
                                   U32       ulInterrupt,
                                   U32       *ulError);
```

Returns – N/A.

Where:

BoardNumber – Defines board number to be used by the driver for a particular device.

userHandle – User supplied handle for the event.

ulInterrupt – The desired interrupt to attach to.

ulError – Returns 0 or error code. Refer to tools.h for a list of error codes.

4.6.14 ADADIO_Trigger_Burst

Triggers a single conversion of all active analog input channels.

Prototype:

```
void  ADADIO_Close_DMA_Channel  (U32     BoardNumber,
                                 U32     *ulError);
```

Returns – N/A.

Where:

BoardNumber – Defines board number to be used by the driver for a particular device.

ulError – Returns 0 or error code. Refer to tools.h for a list of error codes.

4.6.15 ADADIO_SetCallback_Event

Sets a user supplied function to be called when an interrupt occurs. The user must take care to ensure the function can complete before the next interrupt occurs to avoid possible loss of data (FIFO full) or missing events. A sample use is provided in the Analog Inputs function of the example program. Pass a NULL pointer to disable callback events.

Prototype:

U32 ADADIO_SetCallback_Event (**U32** BoardNumber,
EVENT_CB pFunction);

Returns – 0 on success, error code otherwise.

Where:

BoardNumber – Defines board number to be used by the driver for a particular device.

pFunction – User defined function to call.

5. Driver Installation

This section details driver installation on the target system. Any current driver previously installed for the ADADIO must be uninstalled prior to this installation to avoid interference.

To install the driver, API, and associated example files, insert the CD ROM into the drive and close the bay. The installation should commence automatically and display user prompts. Follow the onscreen instructions to complete the installation.

Should the installation fail to automatically start, Select ***Start → Run → Browse*** on the Windows toolbar/popup and browse to find ***Setup.exe*** on the CD ROM. Click on ***OK*** to commence the installation.

The following files are installed on the target system:

OS dependent\…\PciADADIO.sys
OS dependent\…\PlxApi.dll
Program Files\General Standards\ADADIO C\Example.exe
Program Files\General Standards\ADADIO C \ADADIO C Driver.dll
Program Files\General Standards\ADADIO C \ADADIO C Driver.lib
Program Files\General Standards\ADADIO C \ADADIOintface.h
Program Files\General Standards\ADADIO C \ADADIO Example.c
Program Files\General Standards\ADADIO C \Tools.c
Program Files\General Standards\ADADIO C \Tools.h
Program Files\General Standards\ADADIO C \CioColor.h
Program Files\General Standards\ADADIO C \ReadMe.txt
Program Files\General Standards\ADADIO C \ADADIO C.inf

6. Example Program

This section describes the example program, and the files required to develop an application.

The complied example program allows the user to exercise the installed device, while observing the inputs / outputs. To execute, double click on 'Example.exe'. Refer to the Driver Installation section for file location.

The source is provided to educate the user with the GSC API function calls and provide a working example to aid the user with application development. To build the example program using MS Visual C++, create a project and add the following files:

Source Files	→ ***ADADIO Example.c***
	→ ***Tools.c***
Header Files	→ ***ADADIOintface.h***
	→ ***CioColor.h***
	→ ***Tools.h***
Resource Files	→ ***ADADIO C Driver.lib***

Select ***Build → [ProjectName].exe*** on the toolbar.
NOTE: ADADIO C Driver.dll must be in the project directory to run the example.

Contact GSC for example programs (drivers) for other development environments (i.e LabVIEW™, LabWindows/CVI™, etc.)