16AO20

20-Channel 16-Bit High-Speed Analog Output

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

Manual Revision: April 14, 2005

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 16AO20 Windows Driver API developed by General Standards Corporation (GSC). This software provides the interface between the “Application Software” and the 16AO20 board.

The 16AO20 Driver API Software executes under control of the Windows Operating System. The 16AO20 is implemented as a standard Windows driver API written in “C” programming language. The 16AO20 Driver API Software is designed to operate on CPU boards containing x86 processors.

The 16AO20 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 PC104-16AO-20 board contains twenty 16-bit D/A converters (DAC's), and all supporting functions necessary for adding precision high-speed analog output capability to a PCI host. The board is functionally compatible with the IEEE PCI local bus specification Revision 2.1, and supports the "plug-n-play" initialization concept. Unique FIFO buffer controls support the seamless sequencing of successive waveforms. In less demanding applications, the outputs can be updated individually.

A PCI interface adapter provides the interface between the controlling PCI bus and the internal local controller through a 16-bit local bus. Twenty analog output channels are controlled through an analog output FIFO buffer, and can be updated either simultaneously or sequentially. The output sample rate can be controlled by an internal rate generator, or by an external clock. The local controller manages all input/output configuration and data manipulation functions, including auto calibration. Analog output levels are initialized to zero (midrange). Multiboard synchronization is supported.

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 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 50-pin, dual-ribbon front-access 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 16AO20 board:

- PMC-16AO20 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 16AO20 GSC API. The 16AO20 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 16AO20 Win Driver provides an interface to a 16AO20 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 16AO20 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 16AO20 card.
- Readlocal32() - Reads local registers from one 16AO20 card.
- Writelocal32() - Writes to local Registers of one 16AO20 card.
- Close Handle() - Closes a driver interface to one 16AO20 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 AO20_FindBoards()

Detects all PLX Devices connected via the PCI Bus.

Prototype:

```
U32  AO20_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 AO20_Get_Handle

Initializes Handle for the passed board number IN THE DRIVER.

Prototype:

```
U32  AO20_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 AO20_Read_Local32

Read a value from the board local register.

Prototype:

```
U32  AO20_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 AO20intface.h

```
BCR                 Board Control Register
CHAN_SELECT         Channel Select Register
SAMP_RATE           Sample Rate Register
BUFFER_OPS          Buffer Operations Register
FW_REV              Firmware Register
Reserved1           Reserved - Undocumented
OUTPUT_DATA_BUFFER  Output FIFO Register
ADJ_CLOCK           Adjustable Clock Register
```

4.4 AO20_Write_Local32

Write a value to the board local register.

Prototype:

```
void  AO20_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 AOintface.h

```
BCR                 Board Control Register
CHAN_SELECT         Channel Select Register
SAMP_RATE           Sample Rate Register
BUFFER_OPS          Buffer Operations Register
FW_REV              Firmware Register
Reserved1           Reserved - Undocumented
OUTPUT_DATA_BUFFER  Output FIFO Register
ADJ_CLOCK           Adjustable Clock Register
```

uiValue – Value to write to the selected register.
Refer to the 16AO20 user manual for all register / bit definitions.

4.5 AO20_Close_Handle

Closes the device handle and frees the resources.

Prototype:

```
void  AO20_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 AO20_Initialize

Perform a reset on the board. All register values are set to defaults.

Prototype:

```
void  AO20_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 AO20_Autocal

Perform an auto calibration on the board. This operation generates new calibration correction values.

Prototype:

```
U32  AO20_Autocal      (U32      BoardNumber,
                        U32      *ulError);
```

Returns – 0xAA for interrupt timeout, otherwise autocal status (0 or 1)

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 AO20_Set_Output_Mode

Sets the output mode of the board, burst or continuous.

Prototype:

```
void  AO20_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.

ulOuputMode – Valid values: BURST_MODE (1) or CONT_MODE (0).

4.6.4 AO20_Set_Clocking_Mode

Sets the clocking mode of the board outputs, simultaneous or sequential.

Prototype:

```
void  AO20_Set_Clocking_Mode    (U32    BoardNumber,
                                 U32    *ulError
                                 U32    ulClockMode);
```

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.

ulOuputMode – Valid values: SIM_CLOCKING (1) or SEQ_CLOCKING (0).

4.6.5 AO20_Clear_Buffer

Clears all data from the active output buffer.

Prototype:

```
void  AO20_Clear_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 AO20_Sel_Channels

Sets active channels in the group. Channels are numbered 0-19.

Prototype:

```
void  AO20_Sel_Channels     (U32     BoardNumber,
                             U32     ulChannels,
                             U32     *ulError);
```

Returns – N/A

Where:

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

ulChannels – Set the binary bit position of the channel number = 1 to activate the channel, = 0 to disable the channel. Bit0 = Channel0, Bit1 = Channel1, etc.
For example, to activate channel 2, pass 0x04 (100b). Valid values 0 – 0xFFFF.

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

NOTE: Active channels define the group.

4.6.7 AO20_Set_Samp_Rate

Sets the sample rate for the group. Calculations are based on the internal 30MHz clock rate generator if the CLK INITIATOR bit is NOT set (Bit 9 of the ADJ_CLOCK register = 0), or the adjustable rate clock if the CLK_INITIATOR bit is set (=1). The user should call AO20_Set_Adj_Clock_Rate() prior to setting the sampling rate if the adjustable clock is used.

NOTE: If using an external clock, and this board is a target, the user should NOT use this function, but instead write directly to the SAMP_RATE register to set the desired sampling rate based on the external clock frequency.

NOTE: The adjustable clock is optional, and may not be present on the board.

Prototype:

```
double  AO20_Set_Samp_Rate (U32        BoardNumber,
                            double     fRate,
                            U32        *ulError);
```

Returns – Actual sample rate in Hz.

Where:

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

fRate – The desired sample rate (Hz), valid for 457.77 – 400,000.00 Hz.

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

NOTE: The function will coerce values outside of the valid range.

4.6.8 AO20_Set_Adj_Clock_Rate

Sets the adjustable clock generator frequency for the group.
NOTE: The adjustable clock is optional, and may not be present on the board.

Prototype:

```
double  AO20_Set_Adj_Clock_Rate   (U32       BoardNumber,
                                   double    fRate,
                                   U32       *ulError);
```

Returns – Actual clock frequency in Hz (if option present), else 30MHz.

Where:

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

fRate – The desired clock frequency (Hz), valid for 16.0 – 32.0 MHz.

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

NOTE: The function will coerce values outside of the valid range.

4.6.9 AO20_EnableInterrupt

Enables the desired interrupt in the local register, and for the PCI bus. See 16AO20 User manual for interrupt sources.

Prototype:

```
U32  AO20_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.

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

4.6.10 AO20_DisableInterrupt

Disables all interrupts in the local register, and for the PCI bus.

Prototype:

```
void  AO20_DisableInterrupt    (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.

4.6.11 AO20_AttachInterrupt

Attaches a user supplied handle to the associated interrupt event.

Prototype:

```
void  AO20_Attach_Interrupt   (U32       BoardNumber,
                               HANDLE    *userHandle,
                               U32       ulLocalIntr,
                               U32       *ulError);
```

Returns – N/A.

Where:

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

userHandle – User defined handle for event notification.

ulLocalIntr – Local Interrupt to enable and attach the handle to. Valid for 0 – 0x7.

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

4.6.12 AO20_ReAttach_Interrupt

Reattaches a user supplied handle to the associated interrupt event for use in loops.

Prototype:

```
void  AO20_ReAttach_Interrupt     (U32        BoardNumber,
                                   HANDLE     *userHandle,
                                   U32        *ulError);
```

Returns – N/A.

Where:

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

userHandle – User defined handle for event notification.

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

4.6.13 AO20_Open_DMA_Channel

Opens the desired DMA channel for transferring data to the board output FIFO.

Prototype:

```
void  AO20_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, valid for channel 0 or 1.

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

4.6.14 AO20_DMA_Transfer

Transfers the desired number of WORDS to the board output active buffer. This function does not check the active buffer size.

Prototype:

```
U32  AO20_DMA_Transfer      (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, valid for channel 0 or 1.

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.

4.6.15 AO20_Close_DMA_Channel

Closes the desired DMA channel.

Prototype:

```
void  AO20_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, valid for channel 0 or 1.

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

4.6.16 AO20_Enable_Clock

Enables the sampling rate clock to transfer data to the board outputs.

Prototype:

```
void  AO20_Enable_Clock     (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.

4.6.17 AO20_Disable_Clock

Disables the sampling rate clock to stop data transfer to the board outputs.

Prototype:

```
void  AO20_Disable_Clock     (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.

5. Driver Installation

This section details driver installation on the target system. Any current driver previously installed for the 16AO20 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\…\Pci16AO20.sys
OS dependent\…\GSApi.dll
Program Files\General Standards\16AO20\Example.exe
Program Files\General Standards\16AO20\AO Driver.dll
Program Files\General Standards\16AO20\AO Driver.lib
Program Files\General Standards\16AO20\AO20intface.h
Program Files\General Standards\16AO20\16AO20Example.c
Program Files\General Standards\16AO20\Tools.c
Program Files\General Standards\16AO20\Tools.h
Program Files\General Standards\16AO20\CioColor.h
Program Files\General Standards\16AO20\AO20driver.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 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	→ ***16AO20Example.c***
	→ ***Tools.c***
Header Files	→ ***AO20intface.h***
	→ ***CioColor.h***
	→ ***Tools.h***
Resource Files	→ ***AO20 Driver.lib***

Select ***Build → [ProjectName].exe*** on the toolbar.
NOTE: AO20 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.)