# *GX1642*
# *GX1648*
# *GX1649*

## Analog Outputs Boards and GXAO Software

### *User's Guide*

Last updated: March 17, 2016

MARVIN TEST SOLUTIONS

## Safety and Handling

Each product shipped by Marvin Test Solutions is carefully inspected and tested prior to shipping. The shipping box provides protection during shipment, and can be used for storage of both the hardware and the software when they are not in use.

The circuit boards are extremely delicate and require care in handling and installation. Do not remove the boards from their protective plastic coverings or from the shipping box until you are ready to install the boards into your computer.

If a board is removed from the computer for any reason, be sure to store it in its original shipping box. Do not store boards on top of workbenches or other areas where they might be susceptible to damage or exposure to strong electromagnetic or electrostatic fields. Store circuit boards in protective anti-electrostatic wrapping and away from electromagnetic fields.

Be sure to make a single copy of the software CD for installation. Store the original CD in a safe place away from electromagnetic or electrostatic fields. Return compact disks (CD) to their protective case or sleeve and store in the original shipping box or other suitable location.

## Warranty

Marvin Test Solutions products are warranted against defects in materials and workmanship for a period of 12 months. Marvin Test Solutions shall repair or replace (at its discretion) any defective product during the stated warranty period. The software warranty includes any revisions or new versions released during the warranty period. Revisions and new versions may be covered by a software support agreement. If you need to return a board, please contact Marvin Test Solutions Customer Technical Services Department via http://www.marvintest.com/magic/ - the Marvin Test Solutions on-line support system.

## If You Need Help

Visit our web site at http://www.marvintest.com for more information about Marvin Test Solutions products, services and support options. Our web site contains sections describing support options and application notes, as well as a download area for downloading patches, example, patches and new or revised instrument drivers. To submit a support issue including suggestion, bug report or questions please use the following link: http://www.marvintest.com/magic/

You can also use Marvin Test Solutions technical support phone line (949) 263-2222. This service is available between 8:30 AM and 5:30 PM Pacific Standard Time.

## Disclaimer

In no event shall Marvin Test Solutions or any of its representatives be liable for any consequential damages whatsoever (including unlimited damages for loss of business profits, business interruption, loss of business information, or any other losses) arising out of the use of or inability to use this product, even if Marvin Test Solutions has been advised of the possibility for such damages.

## Copyright

## Trademarks

| | |
|---|---|
| ATEasy®, CalEasy, DIOEasy®, DtifEasy, WaveEasy | Marvin Test Solutions<br>(prior name is Geotest - Marvin Test Systems Inc.) |
| C++ Builder, Delphi | Embarcadero Technologies Inc. |
| LabView, LabWindows™/CVI | National Instruments |
| Microsoft Developer Studio, Microsoft Visual C++, Microsoft Visual Basic,, .NET, Windows 95, 98, NT, ME, 2000, XP, VISTA, Windows 7, 8 and 10 | Microsoft Corporation |

All other trademarks are the property of their respective owners.

# Table of Contents

# Chapter 1 - Introduction

## Manual Scope and Organization

### Manual Scope

The purpose of this manual is to provide all the necessary information to install, use, and maintain the GX1642/GX1649 and GX1649 instruments (referred in this manual as GX164x). This manual assumes the reader has a general knowledge of PC based computers, Windows operating systems, and some understanding of digital to analog conversion.

This manual also provides programming information using the GXAO driver. Therefore, good understanding of programming development tools and languages may be necessary.

### Manual Organization

This manual is organized in the following manner:

| Chapter | Content |
|---|---|
| Chapter 1 - Introduction | Introduces the GX164x manual. Lists all the supported board and shows warning conventions used in the manual. |
| Chapter 2 – Overview | Describes the GX164x board's features, board description, its architecture, specifications and the panel description and operation. |
| Chapter 3 – Installation and Connections | Provides instructions on how to install the GX164x boards and its accompanying GXAO software. |
| Chapter 4 – Programming the Board | Provides a listing of GXAO driver files, general purpose/generic driver functions, and programming methods. Discusses various supported operating systems and development tools. |
| Chapter 5 – Functions Reference | Contains a listing of the the GXAO functions. Each function is described along with its syntax, parameters, and special programming comments. Samples are given for each function. |

## Conventions Used in this Manual

| Symbol Convention | Meaning |
|---|---|
|  | Static Sensitive Electronic Devices. Handle Carefully. |
|   | Warnings that may pose a personal danger to your health. For example, shock hazard. |
|  | Cautions where computer components may be damaged if not handled carefully. |
|   | Tips that aid you in your work. |

| **Formatting Convention** | **Meaning** |
|---|---|
| `Monospaced Text` | Examples of field syntax and programming samples. |
| **Bold type** | Words or characters you type as the manual instructs, programming function names and window control names. |
| *Italic type* | Specialized terms.<br>Titles of other reference books.<br>Placeholders for items you must supply, such as function parameters |

# Chapter 2 - Overview

## Introduction

The GX1648, GX1642, and the GX1649/GX1649-1 boards are 3U PXI boards and provide precision analog outputs. The boards are configured as four groups, with each group having 16 analog output channels with 12-bit resolution for the GX1642 and GX1648 modules. The GX1649 offers 16 bit resolution. The voltage range for each group can be programmed to 0-10V, ±10V, 0-5V or ±5V for the GX1642. The GX1648 can be programmed with the following values: 0-20V, ±20V, 0-10V or ±10V. The GX1649 supports a single range of ±15V.. Each group's channel voltages can be set simultaneously through software or by an internal or external command signal. The GX1649 also includes an internal memory that can be used to generate waveforms with a sample rate of up to 10 MS/s, enhancing the maximum software-based update rate of 10 KS/s for the GX1642/GX1648. The GX1649-1 support streaming the waveform using user supplied interrupt callback.

## Features

Principal capabilities of the GX1648/GX1642/GX1649/GX1649-1 are summarized in the following list of features.

- 64 precision, individual analog output channels
- 12-Bit D/A converter per channel for the GX1642/48 and 16-bit D/A for the GX1649
- Accuracy of 3mV, ±1LSB for the GX1642/48 and 2mV, ±4LSB for the GX1649
- Each channel is able to source/sink up to 10mA for all voltage settings
- Data update rate of 10 KSPS (Kilo samples per second)
- Outputs are updated simultaneously or per channel (software-selectable)
- Four external group voltage inputs for each group allows for simultaneous continuous update
- External update input supports simultaneous updating of all groups
- Each group can be programmed to one of the following output voltage ranges:
    - GX1648: 0-10V, ±10V, 0-5V or ±5V
    - GX1642: 0-20V, ±20V, 0-10V or ±10V
    - GX1649: ±15V
- Fast settling time of 50uSec to 0.1% of programmed value. The GX1649 supports an update rate of up to 10 MS/s using its on board, DAC and the 256K of sample memory.
- 8 Channel digital I/O with input, output, and direction memories (8 Kbyte) as well as a programmable clock.(GX1649 only)
- All channels are automatically set to zero volts DC after a power reset
- Output TTL pin (GX1642, GX1648 only)
- Upgradeable firmware using in-system-program software driver
- Streaming callback support (GX1649-1)

## Board Description

The GX1642/1648/GX1649 boards' channels are organized as four groups with 16 channels each. The GX1642 / GX1648' channels have 12-bit D/A converters (DAC). The GX1649 has 16-bit D to A converters. Each group can be independently set to one of the four available voltage ranges: 0-10V, ±10V, 0-5V or ±5V for the GX1642. The GX1648 supports 0-20V, ±20V, 0-10V or ±10V ranges. A single range of ±15V is supported for the GX1649. Changing the voltage range (GX1642/GX1648 only) will change the group's channels voltage resolution. For example changing the range from ±10V to 0-5V will change the resolution from 4.88mV to 1.22mV. All channels will source or sink a minimum of 10mA of supply current for all voltage ranges. In addition to static output channels, the GX1649 has 256K of sample memory for each group which can be used to generate waveforms. Each of the four channel groups can be configured as a waveform generator or as static outputs.

The board has a local controller that performs all internal configuration and data manipulation between the PXI bus and the board. Values are first loaded to the specified channel output buffer before being programmed to the corresponding DAC. Each DAC has an output buffer enabling users to load all (or part) of the board's channels with new values and then updating all outputs at once.

The local controller reads the 16-bit channel data value for each channel from the analog output buffer, and sends the value serially to the associated output DAC. The output DAC holds this value in an internal buffer until commanded to transfer the data to the output register that drives the DAC output. All output registers are updated simultaneously.

Each group has a dedicated external Update Line available at the 78-pin connector (J3/J6). A ground or TTL low on the External Update Line will cause the group's channels to continuously update the group outputs if the group's external update function is enabled through the API (**GxAoSetGroupExternalUpdate**).

For the GX1649, the External Update Lines are driven by GPIO0-GPIO3.The Gx1649 also uses GPIO0-GPIO3 as external trigger inputs for the Group A-D ARBs.

A 78-pin D-type connector on the front panel supports the following signals:

- 64 Output Channels
- Four Group Update external input signals
- Update all groups external input signal
- TTL output signal
- Ground signals

### GPIO Channels (Gx1649 only)

The GX1649's 8 GPIO channels (GPIO0-GPIO7) are general purpose TLL IO that can be configured as inputs (tri-stated) or outputs on a per channel basis using the API (**GxAoDioSetOutputEnable**).

GPIO0-GPIO3 signals are also used as inputs for each group's external update and external ARB trigger functions.

All 8 GPIO channels can also be used by the dynamic digital sequencer to output vectors as well as record them into memory (8192 8-bit vectors). See the function reference for more information regarding the **GxAoDioxxx** functions.

### External ARB Triggers (Gx1649 only)

Each ARB Group (A-D) has a dedicated external trigger pin (GPIO0-GPIO3). Each group can be independently configured to use its dedicated external trigger pin to begin ARB sequencing (**GxAoArbSetGroupTrigger**). An ARB Group must be armed before it can act on an external trigger event (**GxAoArbArmGroup**).

If an external signal is used as a trigger source, the GPIO channel's output should first be disabled (tri-stated) using the API (**GxAoDioSetOutputEnable**).

The external trigger using the GPIO is detected on a rising edge.

Each ARB Group can also be configured to use the PXI0-PXI7 or STAR backplane triggers.

### ARB Streaming Mode (Gx1649-1 only)

Each ARB Group (A-D) can operate in streaming mode (**GxAoArbEnableGroupStreaming**) where the ARB samples are read from a 1K sample FIFO instead of the onboard memory. The FIFO can be written to by the user while the ARB sequencer is running (**GxAoArbWriteStreamingData**). When the FIFO is half empty (less than 512 samples), a PCI interrupt is generated and the user can provide an interrupt handler function to write more samples to the FIFO. Alternatively, the user can poll the streaming status (**GxAoArbGetGroupStreamingStatus**) and write more samples to the FIFO when desired.

Figure 2-1 and Figure 2-2 shows the GX1648 and GX1642 with the J3 connector.

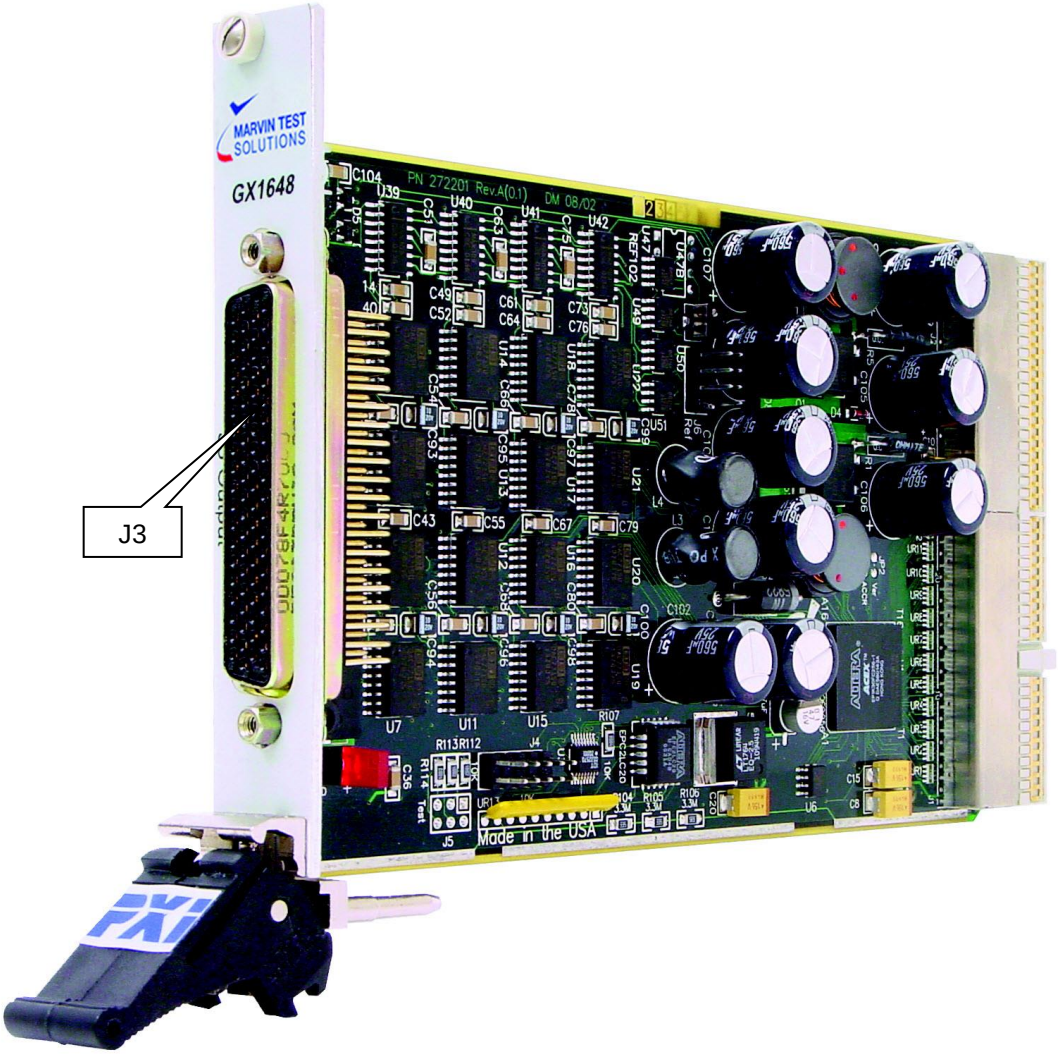


**Figure 2-1: GX1648 Board (side view)**

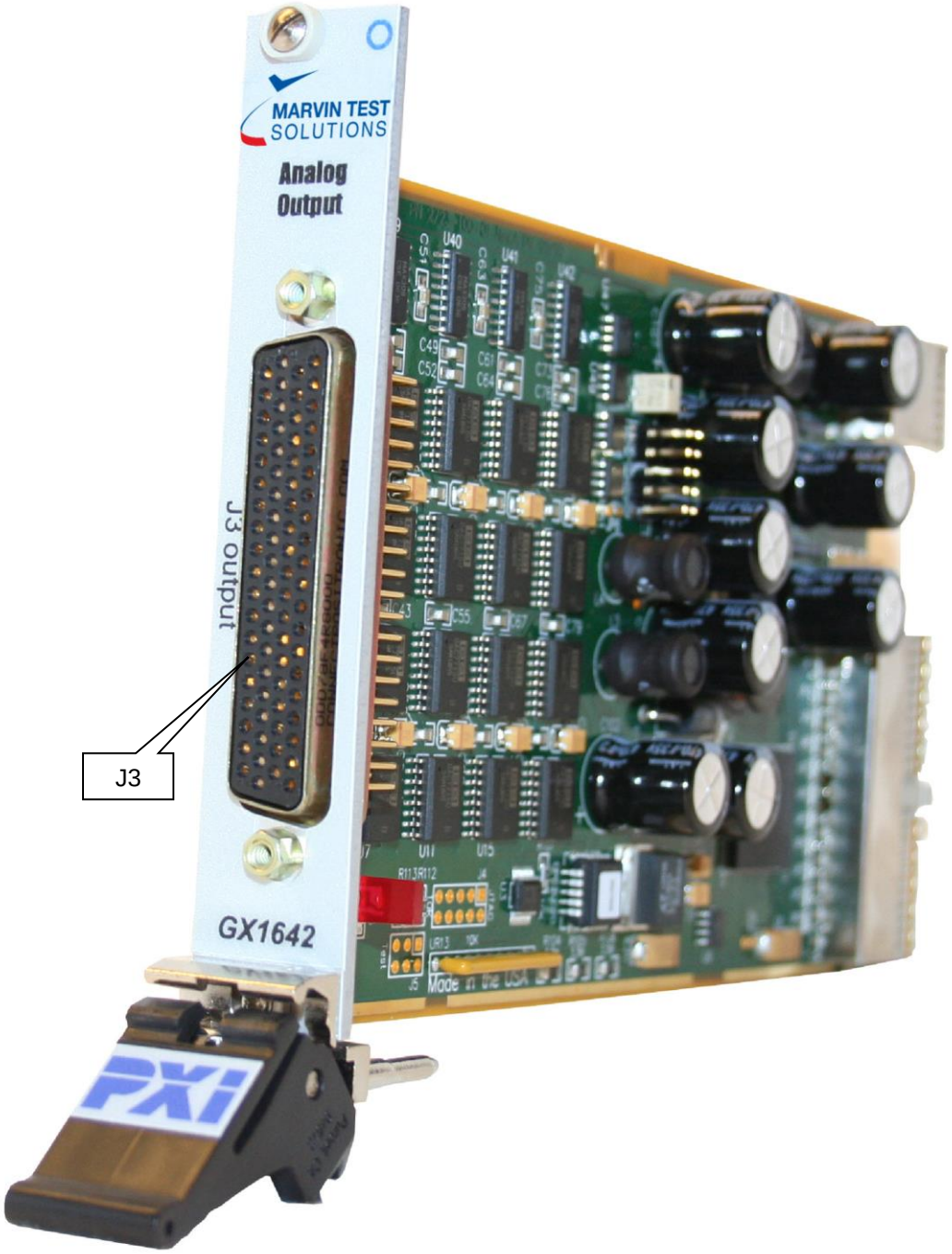


**Figure 2-2: GX1642 Board (side view)**

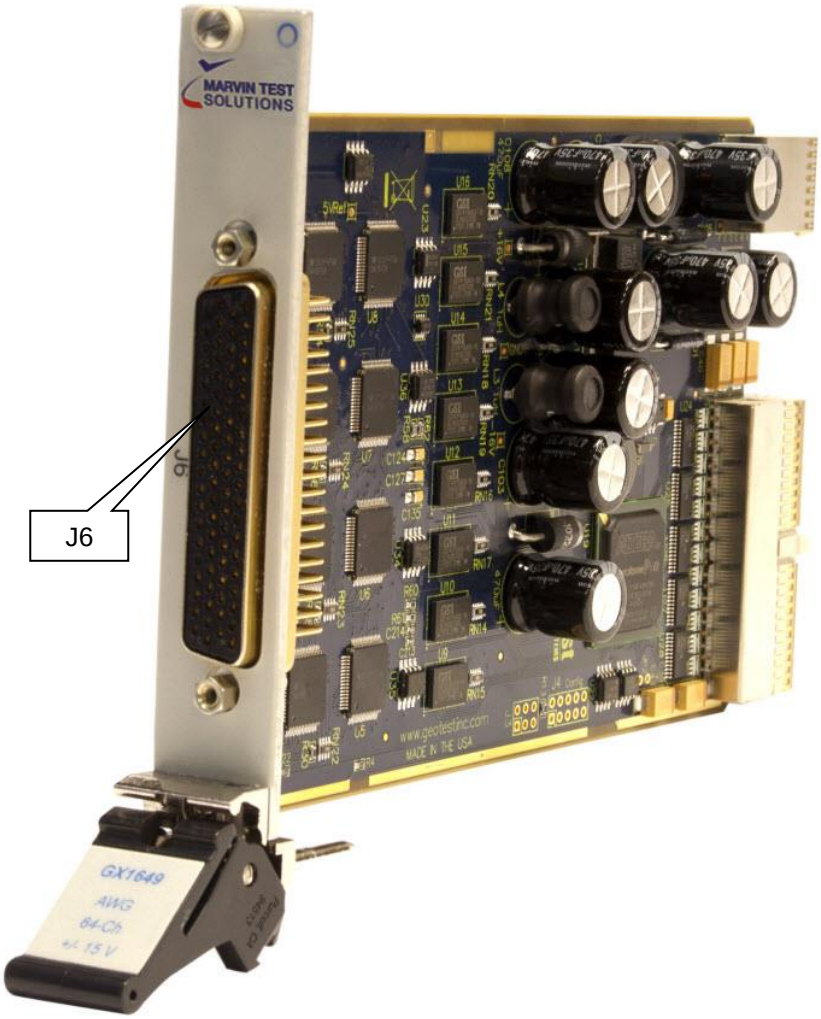


**Figure 2-3: GX1649 Board (side view)**

## Architecture

Figure 2-4 illustrates the GX1642/GX1648 boards' common static voltage architecture. The board is programmed by software using the PXI bus interface to set the group's range, D/A output and update control, and control of the board's 4 digital I/O pins. The 78-pin connector on the right (J3) provides the interface to the D/As' outputs, digital output and input signals, and the external update signals.

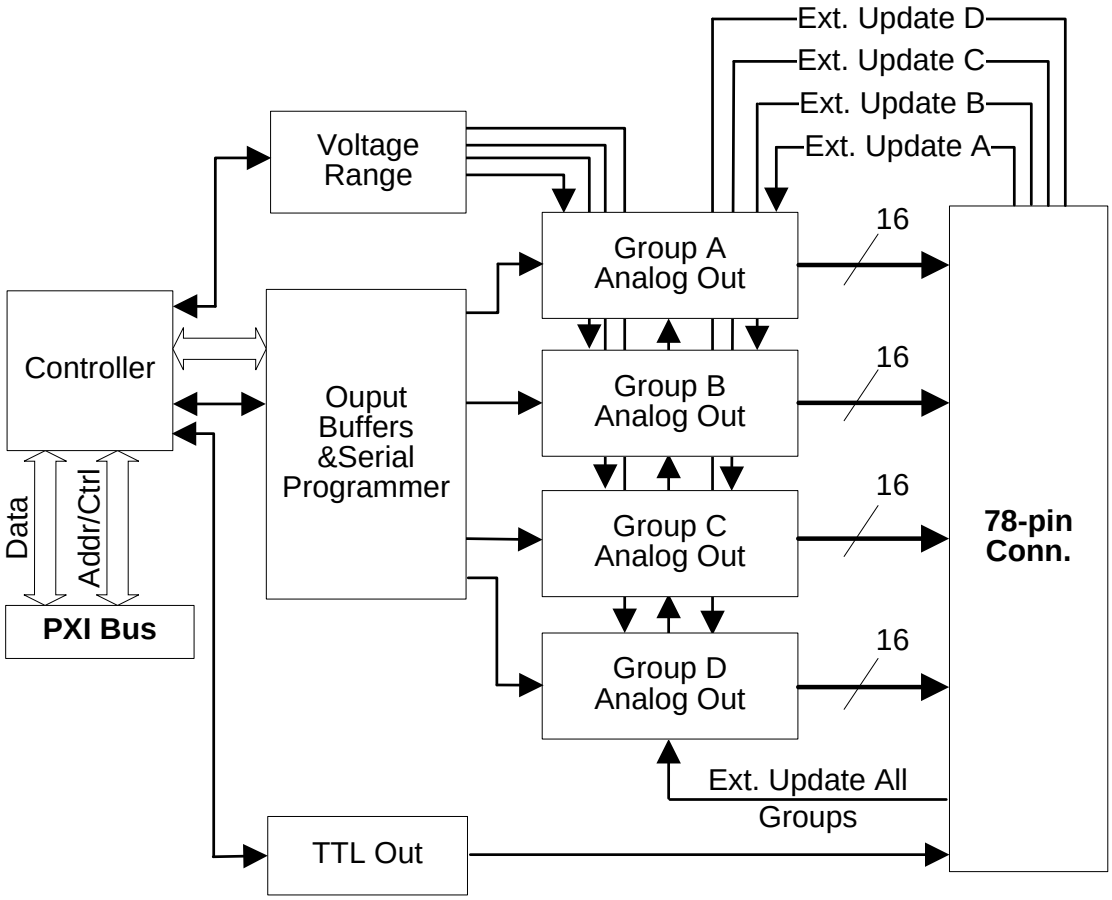


**Figure 2-4: GX1642/GX1648 Block Diagram**

Figure 2-5 illustrates the GX1649 board's architecture. The board includes an ARB circuit and memory for each Group (A to D) as well as a DIO sequencer that interfaces with the 8 TTL I/O Channels. Each of the 64 channels can function has a static analog output or as an ARB.

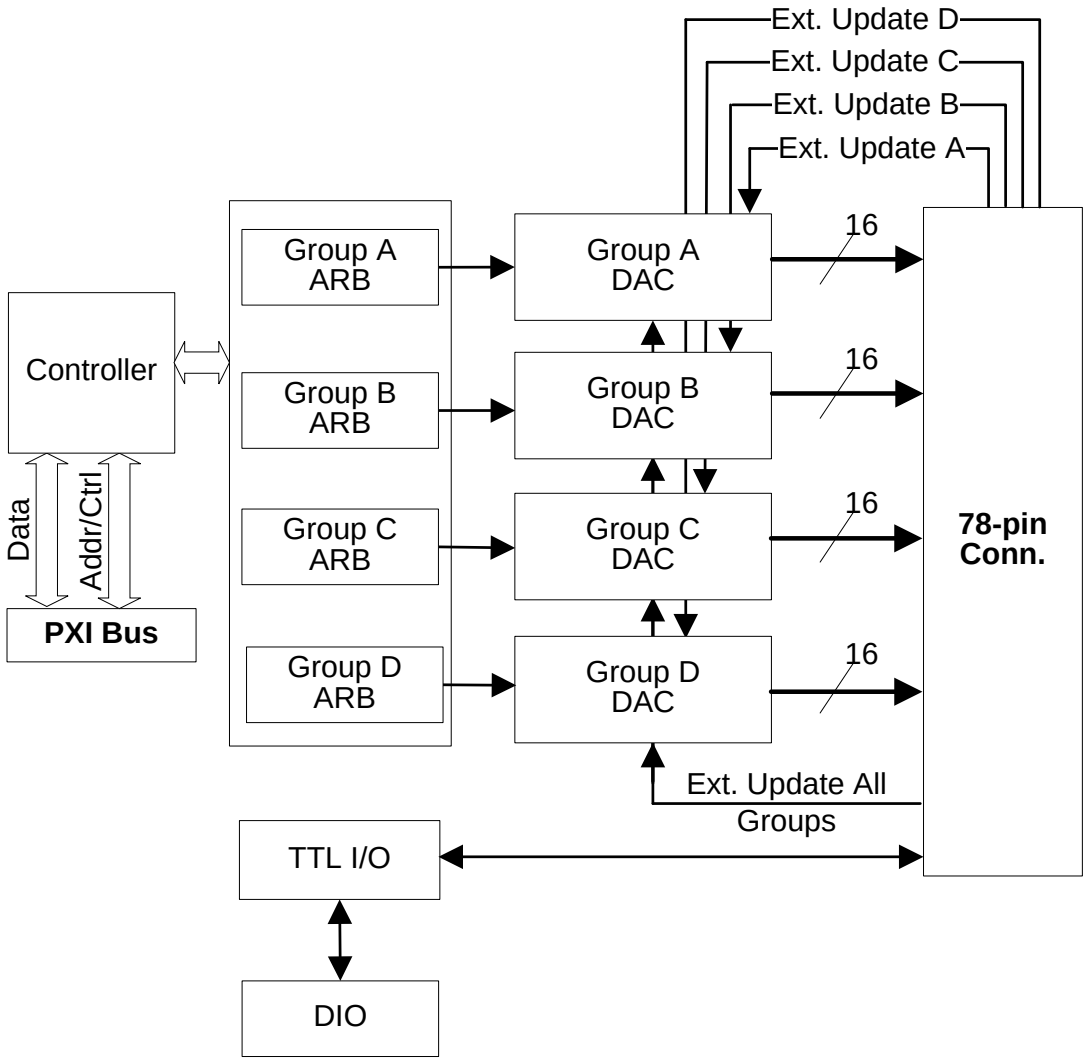


**Figure 2-5: GX1649 Block Diagram**

## Specifications

The following table outlines the specifications of the GX1648/GX1642 boards.

| | |
|---|---|
| Number of Output channels | 64 |
| Output Voltage Ranges: | |
| Number of Ranges | 4 (each group) |
| Selectable Output Voltage Ranges | GX1642: 0-20V, ±20V, 0-10V or ±10V<br>GX1648: 0-10V, ±10V, 0-5V or ±5V |
| Resolution | 12-bit |
| | |
| Accuracy | 3mV ±1 LSB |
| Slew Rate | 0.5V/µs |
| Settling Time | 50 uSec to 0.1% |
| Max. Current per Channel | GX1642: 10 mA<br>GX1648: 10 mA |
| Power Supply Current | 3.3V Rail: 100 mA (max)<br>5V Rail: 2 A (typical), 6 A (max)<br>12 V Rail: 10mA (max) |
| Operating Temperature | 0 to+55°C |
| Storage Temperature | -20 to+70°C |
| Size | 3U PXI Card |
| Weight | 12 oz. |

The following table outlines the specifications of the GX1649 board.

| | |
|---|---|
| General | |
| Number of Output channels | 64 |
| Number of Ranges | 1 (each group) |
| Output Voltage Range | ±15V |
| Resolution | 16-bit |
| | |
| Accuracy | ± 4 LSB, -14.75V to +14.75 V<br>± (4 LSB + 2 mV), < - 14.75 V & > + 14.75 V |
| Slew Rate | 3V/µs |
| Settling time | 50 uSec to 0.1% |
| Max. Current per channel | ± 5mA |
| Power supply Current | 3.3V Rail: 100 mA (max)<br>5V Rail: 2 A (typical), 6 A (max) |

| | |
|---|---|
| | 12 V Rail: 10mA (max) |
| Operating Temperature | 0 to+55°C |
| Storage Temperature | -20 to+70°C |
| Size | 3U PXI Card |
| Weight | 12 oz. |
| | |
| Arbitrary Waveform Generator | |
| Memory | 256K Samples allocated per group of 16 channels. One or all channels can be selected for allocation |
| Sample Rate | 625 KHz per channel if all channels are active |
| Programmable Sample Clock | 10 MHz to 0.5 Hz, one per bank<br>Note: Only one channel per bank can be active for the 10 MHz rate |
| Sample Clock Resolution | 10 MHz / $2^N$, N is 0 to 31 |
| Sample Clock Sources | Internal (10 MHz),<br>External (via digital I/O inputs, one per bank)<br>PXI trigger bus<br>PXI Star trigger bus |
| | |
| Digital IO | |
| Channels | 8, each configurable as input or output.<br>Four of the channels can be used as external inputs for AWG clocks |
| Memory | 8K byte vectors |
| Logic Levels | TTL |

## Virtual Panel Description

The GX164x includes a virtual panel program, which enables full utilization of the various configurations and controlling modes. To fully understand the front panel operation, it is best to become familiar with the functionality of the board.

To open the virtual panel application, select **GxAo Panel** from the **Marvin Test Solutions**, **GXAO** menu under the **Start** menu. The GX1642/8/9 virtual panel opens as shown here:

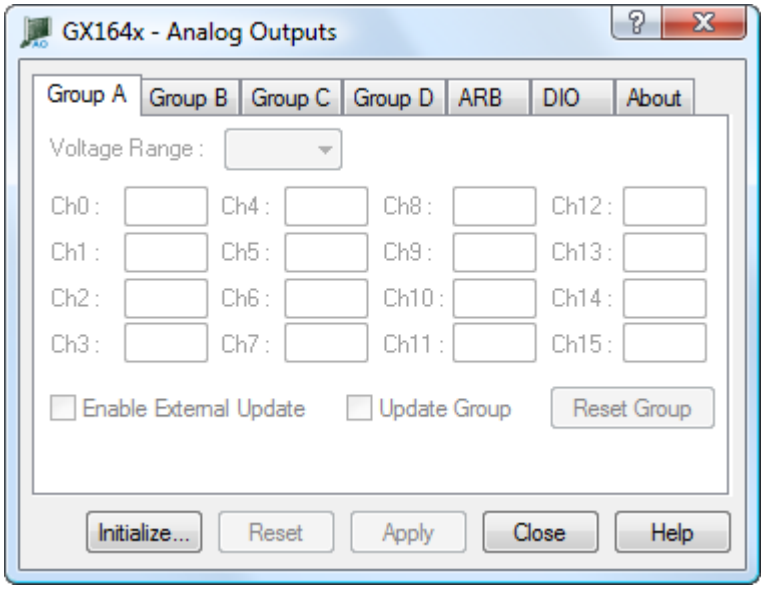


**Figure 2-6: GX164X Virtual Panel (Un-Initialized)**

The function of the panel button controls are shown below:

**Initialize** Opens the Initialize Dialog (see Initialize Dialog paragraph) in order to initialize the board driver. The current settings of the selected counter **will not change after calling initialize**. The panel will reflect the current settings of the counter after the Initialize dialog closes.

**Reset -** resets the PXI board settings to their default state and clears the reading.

**Apply** – applies changed settings to the board

**Close -** closes the panel. Closing the panel **does not affect** the counter settings.

**Help -** opens the on-line help window. In addition to the help menu, the caption shows a **What's This Help** button (?) button. This button can be used to obtain help on any control that is displayed in the panel window. To display the What's This Help information click on the (?) button and then click on the control – a small window will display the information regarding this control.

## Virtual Panel Initialize Dialog

The Initialize dialog initializes the driver for the selected counter board. The counter settings **will not change** after initialize is called. Once initialize, the panel will reflect the current settings of the counter.

The Initialize dialog supports two different device drivers that can be used to access and control the board:

1. **Use Marvin Test Solutions's HW** – this is the device driver installed by the setup program and is the default driver. When selected, the **Slot Number** list displays the available counter boards installed in the system and their slots. The chassis, slots, devices and their resources are also displayed by the HW resource manager, **PXI/PCI Explorer** applet that can be opened from the Windows Control Panel. The PXI/PCI Explorer can be used to configure the system chassis, controllers, slots and devices. The configuration is saved to PXISYS.INI and PXIeSYS.INI located in the Windows folder. These configuration files are also used by VISA. The following figure shows the slot number 0x10D (chassis 1 Slot 13). This is the slot number argument (*nSlot*) passed by the panel when calling the driver **GxAoInitialize** function used to initialize driver with the specified board.

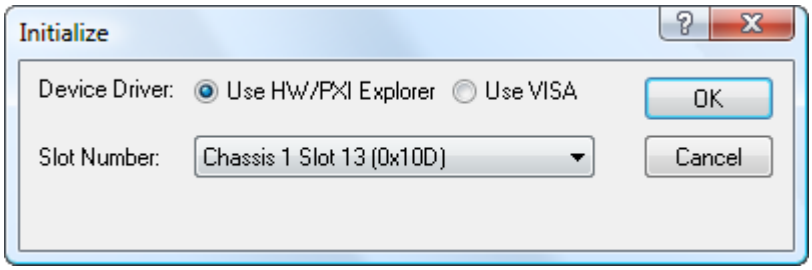


**Figure 2-7: Initialize Dialog Box using Marvin Test Solutions's HW driver**

2. **Use VISA** – this is a third party device driver usually provided by National Instrument (NI-VISA). When selected, the **Resource** list displays the available boards installed in the system and their VISA resource address. The chassis, slots, devices and their resources are also displayed by the VISA resource manager, **Measurement & Automation** (NI-MAX) and in Marvin Test Solutions's **PXI/PCI Explorer**. The following figure shows PXI7::10::INSTR as the VISA resource (PCI bus 7 and Device 10). This is VISA resource string argument (*szVisaResource*) passed by the panel when calling the driver **GxAoInitializeVisa** function to initialize the driver with the specified board.

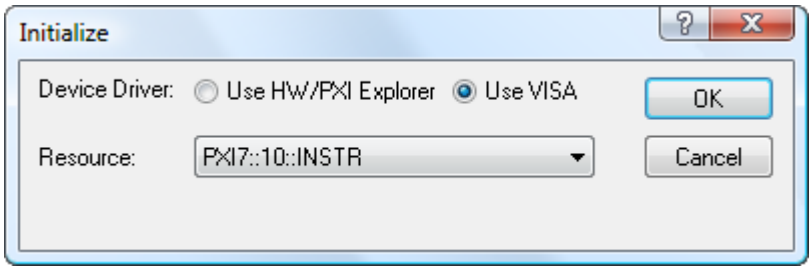


**Figure 2-8: Initialize Dialog Box using VISA resources**

## Virtual Panel Group A/B/C/D Pages

The following picture shows the **Group A page** followed by an explanation of each one of the controls displayed on the page:

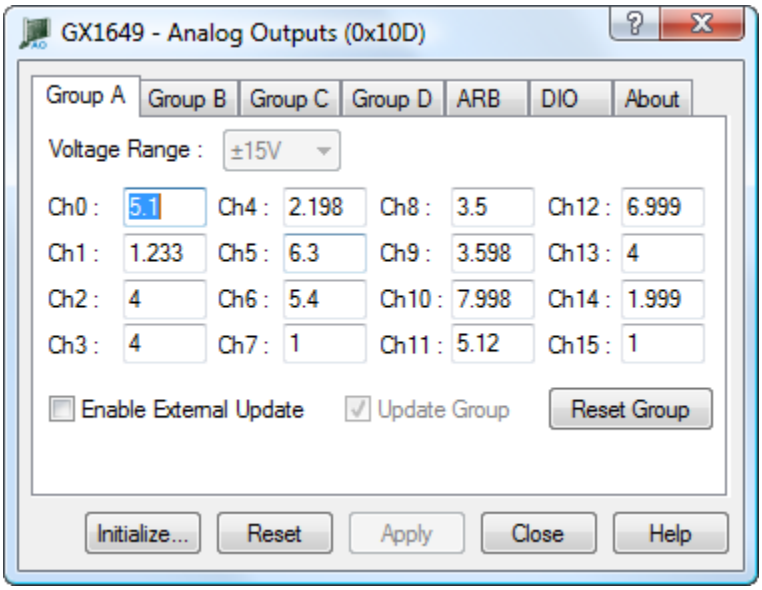


**Figure 2-9: GX164X Virtual Panel – Group A page**

**Voltage Range Dropdown list:** Select/display the group voltage range.

**Ch 0-15 Edit Boxes:** Each channel is displayed with an edit box for displaying the loaded voltage value. The value is updated continuously until the user types a new value to set. Clicking on the **Apply** button (or the **Enter** key) will load the board with the modified values and if the **Update Group** is checked will also output the new DAC values. After the **Apply** was pressed the panel will resume updating and display the board loaded values continuously.

**Update Group Check Box**: When checked (default), clicking on the **Apply** button the group's channels will load and update the DACs to output the new values. When unchecked, clicking on the **Apply** button the group's channels will only be loaded and the DAC output will not change.

**Enable External Update**: When checked the specified group can be updated by the groups' external input line, when unchecked the groups' external input line is disabled.

**Reset Group Button**: Sets all the groups' channels to zero volts, set voltage range to 0-10V and disables the External Update.

## Virtual Panel ARB Page

The following picture shows the **ARB page (GX1649 Only)** followed by an explanation of each one of the controls displayed on the page:

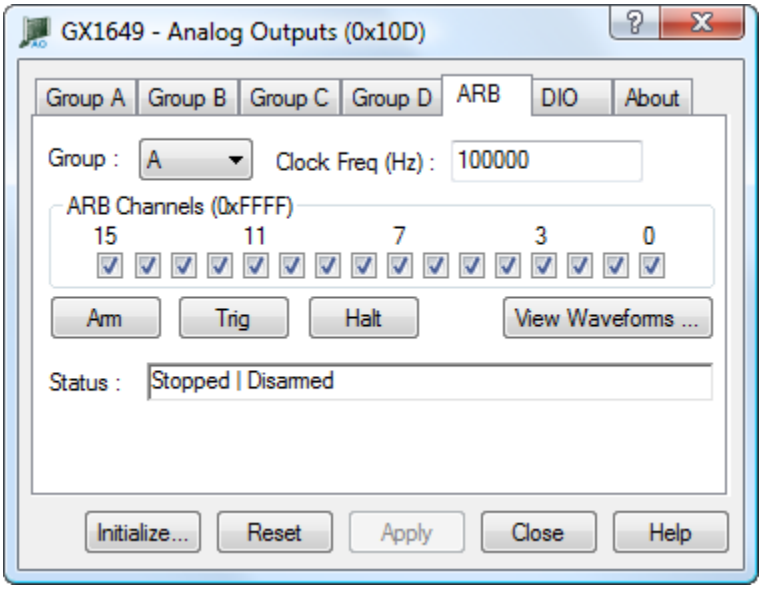


**Figure 2-10: GX1649 (Only) Virtual Panel – ARB page**

**Group Dropdown list:** Select/display the ARB group

**Clock Freq Edit Box:** Used to enter the desired frequency for the ARB Group sequencer

**ARB Channels Group Box:** Displays the hex representation of which channels in the Group are selected for participation in the ARB function. Contains the ARB Channels Checkboxes.

**ARB Channels Checkboxes:** Select/display the channels in the Group which will participate in the ARB function

**Arm button:** Arms the ARB Group sequencer prior to a trigger

**Trig button:** Triggers the ARB Group sequencer (for execution)

**Halt button:** Halts (Disarms) the ARB Group sequencer

**View Waveforms button…:** Displays the waveform stored in ARB memory for the selected Group, see Figure 2-11: ARB page Waveform Dialog Box (GX1649 Only)

**Status label:** Displays the ARB Group sequencer status

Figure 2-11 shows the **ARB page Waveform Dialog Box (GX1649 Only)** followed by an explanation of each one of the controls displayed on the page:

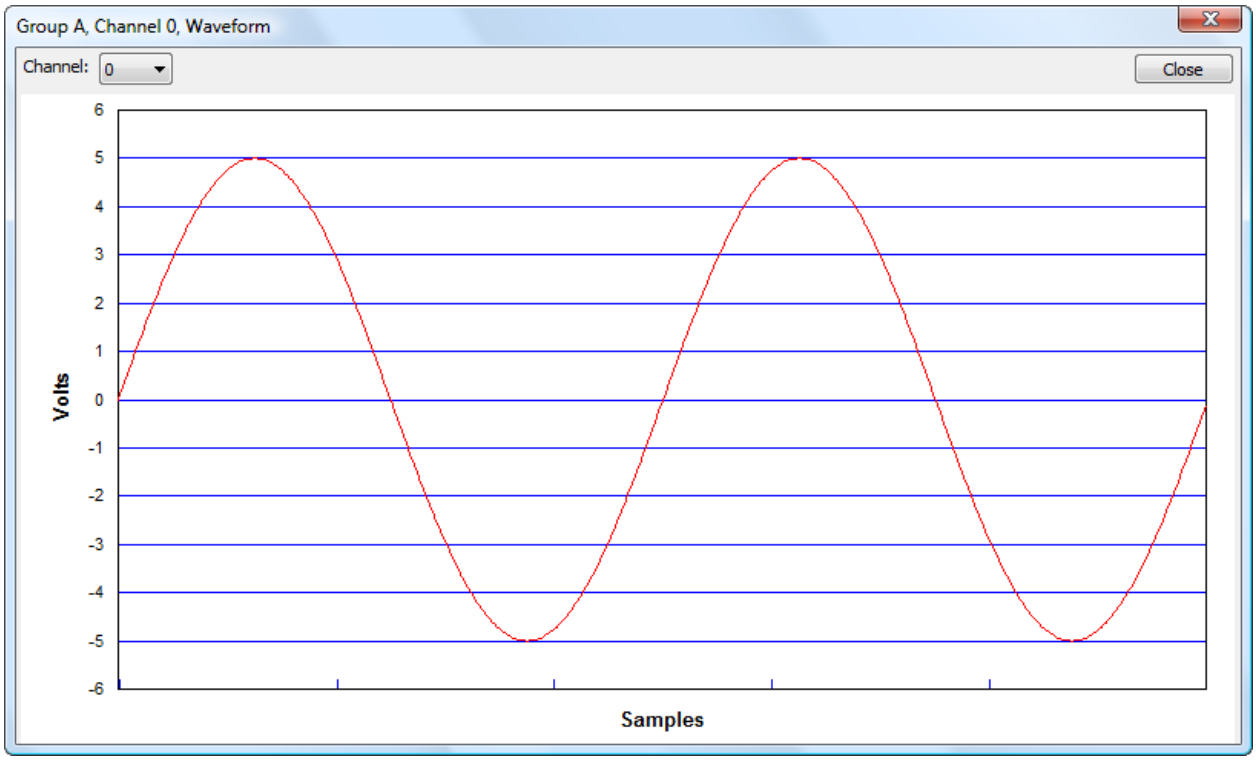


**Figure 2-11: ARB page Waveform Dialog Box (GX1649 Only)**

**Channel Dropdown list:** Select/display the Channel within the currently selected Group to view

## Virtual Panel DIO page

Figure 2-12 shows the **DIO page (GX1649 Only)** followed by an explanation of each one of the controls displayed on the page:

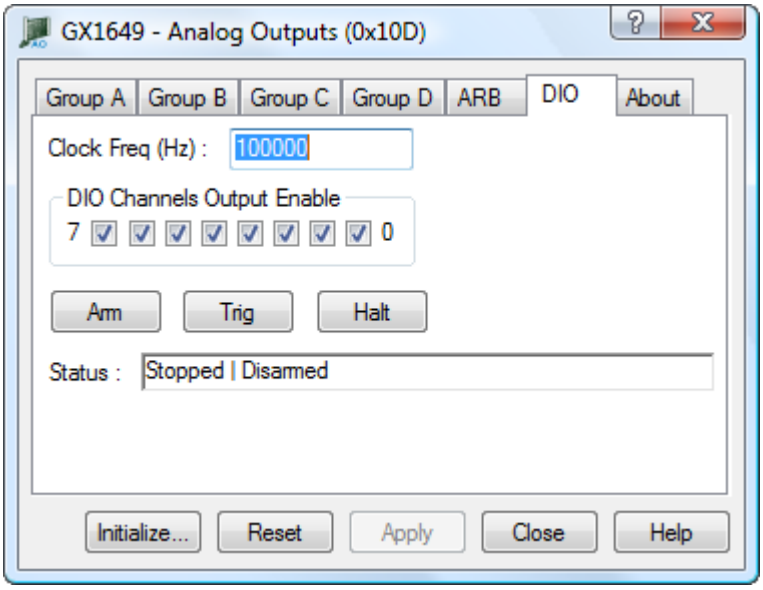


**Figure 2-12: GX1649 (Only) Virtual Panel – DIO page**

**Clock Freq Edit Box:** Used to enter the desired frequency for the DIO sequencer

**DIO Channels Output Enable Checkboxes:** Select/display the DIO channel output enable state

**Arm button:** Arms the DIO sequencer prior to a trigger

**Trig button:** Triggers the DIO sequencer (for execution)

**Halt button:** Halts the DIO sequencer

**Status label:** Displays the DIO sequencer status

## Virtual Panel About Page

Clicking on the **About** tab will show the **About page** as shown in Figure 2-13:

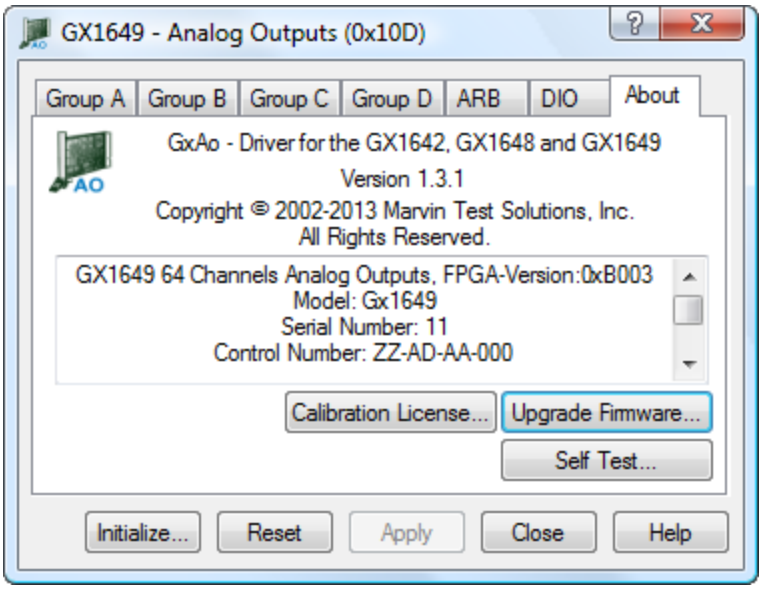


**Figure 2-13: GX164X Virtual Panel – About Page**

The top part of the **About** page displays version and copyright of the GXAO driver. The bottom part displays the board summary, including the EEPROM version, the board Revision, the FPGA version, the board serial number and the calibration time. The **About** page also contains a button **Upgrade Firmware…** used to upgrade the board FPGA. This button maybe used only when the board requires upgrade as directed by Marvin Test Solutions's support. The upgrade requires a firmware file (.jam) that is written to the board FPGA. After the upgrade is complete you must shut down the computer to recycle power to the board in order the new firmware to be used.

The **Self Test…** button starts the board self-test function. Make sure to make sure the J6 front connector is disconnected before running the self-test. The self-test checks the output voltage for each group's channels and runs a memory test on each ARB Memory (Group A to Group D) as well as the DIO memories (Output, Input, and Direction).

# Chapter 3 - Installation and Connections

## Getting Started

This section includes general hardware installation procedures for the GX164x boards and installation instructions for the GXAO software. Before proceeding, please refer to the appropriate chapter to become familiar with the board being installed.

| To Find Information on… | Refer to… |
|---|---|
| Hardware Installation | This Chapter |
| GXAO Driver Installation | This Chapter |
| Programming | Chapter 4 |
| GXAO Function Reference | Chapter 5 |

### Packing List

All GXAO boards have the same basic packing list, which includes:

1. GX1642, GX1648, or GX1649 board
2. GXAO Driver Disk

### Unpacking and Inspection

After removing the board from the shipping carton:

**Caution -** Static sensitive devices are present. Ground yourself to discharge static.

1. Remove the board from the static bag by handling only the metal portions.
2. Be sure to check the contents of the shipping carton to verify that all of the items found in it match the packing list.
3. Inspect the board for possible damage. If there is any sign of damage, return the board immediately. Please refer to the warranty information at the beginning of the manual.

### System Requirements

All GXAO instrument boards are designed for use with a 3U or 6U cPCI or PXI compatible chassis and the software is compatible with any computer system running 32 or 64 bit Windows and Linux (using the GtLinux software package) operating systems.

Each board requires one unoccupied 3U PXI bus slot.

## Installation of the GXAO Software

Before installing the board it is recommended to install the software as described in this section:

1. Insert the Marvin Test Solutions CD-ROM and locate the **GXAO.EXE** setup program. If you computer's Auto Run is configured, when inserting the CD a browser will show several options, select the Marvin Test Solutions Files option, then locate the setup file. If Auto Run is not configured you can open the Windows explorer and locate the setup files (usually located under \Files\Setup folder). You can also download the file from Marvin Test Solutions web site (www.marvintest.com).
2. Run the setup and follow the instruction on the Setup screen to install the software.

   **Note:** You may be required to restart the setup after logging-in as a user with Administrator privileges. This is required in-order to upgrade your system with newer Windows components and to install the HW kernel-mode device drivers that are required by the GXAO driver to access resources on your board.

3. The first setup screen to appear is the Welcome screen. Click **Next** to continue.
4. Enter the folder where the software is to be installed. Either click **Browse** to set up a new folder, or click **Next** to accept the default entry of `C:\Program Files\Marvin Test Solutions\GXAO` or `C:\Program Files (x86)\Marvin Test Solutions\GXAO` for 64 bit Windows.
5. Select the type of Setup you wish and click **Next.** You can choose between **Typical**, **Run-Time** and **Custom** setups. **Typical** setup type installs all files. **Run-Time** setup type will install only the files required for controlling the board either from its driver or from its virtual panel. **Custom** setup type lets you select from the available components.

The program will now start its installation. During the installation, Setup may upgrade some of the Windows shared components and files. The Setup may ask you to reboot after it complete if some of the components it replaced where used by another application during the installation – do so before attempting to use the software.

You can now continue with the installation to install the board. After the board installation is complete you can test your installation by starting a panel program that let you control the board interactively. The panel program can be started by selecting it from the Start, Programs, GXAO menu located in the Windows Taskbar.

## Overview of the GXAO Software

Once the software installed, the following tools and software components are available:

- **PXI/PCI Explorer applet** – use to configure the PXI chassis, controllers and devices. This is required for accurate identification of your PXI instruments later on when installed in your system. The applet configuration is saved to PXISYS.ini and PXIeSYS.ini that are used by Marvin Test Solutions' instruments, the VISA provider and VISA based instruments drivers. In addition, the applet can be used to assign chassis numbers, Legacy Slot numbers and instruments alias names.

  **VISA** is a standard maintained by the VXI Plug & Play System Alliance and the PXI Systems Alliance organizations (http://www.vxipnp.org/, http://www.pxisa.org/). VISA provides a standard way for instrument manufacturers and users to write and use instruments drivers. The VISA resource managers such as National Instruments **Measurement & Automation** (NI-MAX) can display and configure instruments and their address (similar to Marvin Test Solutions' PXI/PCI Explorer).

- **GXAO Panel** – use to configure, control and display the board settings and counter reading.
- **GXAO driver** - A DLL based function library (GXAO.DLL or GXAO64.DLL, located in the Windows System folder) used to program and control the board. The driver uses Marvin Test Solutions' HW driver or VISA supplied by third party vendor to access and control the GXAO board.
- **Programming files and examples** – interface files and libraries for various programming tools, see later in this chapter for a complete list of files, programming languages and development tools supported by the driver.
- **Documentation** – On-Line help and User's Guide.

## Configuring Your PXI System using the PXI/PCI Explorer

To configure your PXI/PCI system using the **PXI/PCI Explorer** applet follow these steps:

1. **Start the PXI/PCI Explorer applet**. The applet can be start from the Windows Control Panel or from the Windows Start Menu, **Marvin Test Solutions**, **HW**, **PXI/PCI Explorer**.
2. **Identify Chassis and Controllers**. After the PXI/PCI Explorer started it will scan your system for changes and will display the current configuration. The PXI/PCI Explorer automatically detects systems that have Marvin Test Solutions controllers and chassis. In addition, the applet detects PXI-MXI-3/4 extenders in your system (manufactured by National Instruments). If your chassis is not shown in the explorer main window, use the Identify Chassis/Controller commands to identify your system. Chassis and Controller manufacturers should provide INI and driver files for their chassis and controllers to be used by these commands.
3. **Change chassis numbers, PXI devices Legacy Slot numbering and PXI devices Alias names.** These are optional steps to be performed if you would like your chassis to have different numbers. Legacy slots numbers are used by older Marvin Test Solutions or VISA drivers. Alias names can provide a way to address a PXI device using your logical name (e.g. "DMM1"). For more information regarding these numbers see the **GxAoInitialize** and **GxAoInitializeVisa** functions.
4. **Save you work**. PXI Explorer saves the configuration to the following files located in the Windows folder: PXISYS.ini, PXIeSYS.ini and GxPxiSys.ini. Click on the **Save** button to save you changes. The PXI/Explorer prompt you to save the changes if changes were made or detected (an asterisk sign ' *' in the caption indicated changes).

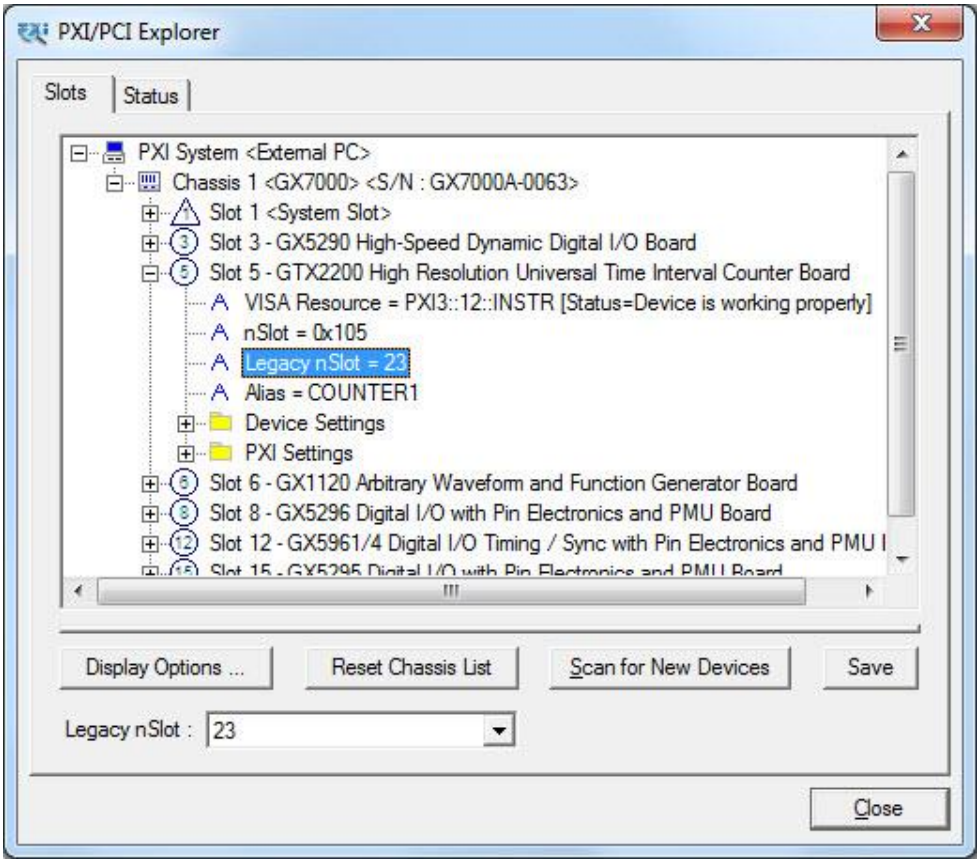


**Figure 3-1: PXI/PCI Explorer**

## Board Installation

### Before you Begin

- Install the software driver as described in the prior section.
- Configure your PXI/PC system using **PXI/PCI Explorer** as described in the prior section.
- Verify that all the components listed in the packing list (see previous paragraph) are present.

### Electric Static Discharge (ESD) Precautions

To reduce the risk of damage to the board, the following precautions should be observed:

- Leave the board in the anti-static bags until installation requires removal. The anti-static bag protects the board from harmful static electricity.
- Save the anti-static bag in case the board is removed from the computer in the future.
- Carefully unpack and install the board. Do not drop or handle the board roughly.
- Handle the board by the edges. Avoid contact with any components on the circuit board.

**Caution -** Do not insert or remove any board while the computer is on. Turn off the power from the PXI chassis before installation.

### Installing a Board

Install the board as follows:

1. Turn off the PXI chassis and unplug the power cord.
2. Locate a PXI empty slot on the PXI chassis.
3. Place the module edges into the PXI chassis rails (top and bottom).
4. Carefully slide the PXI board to the rear of the chassis, make sure that the ejector handles are pushed **out** (as shown in Figure 3-2).

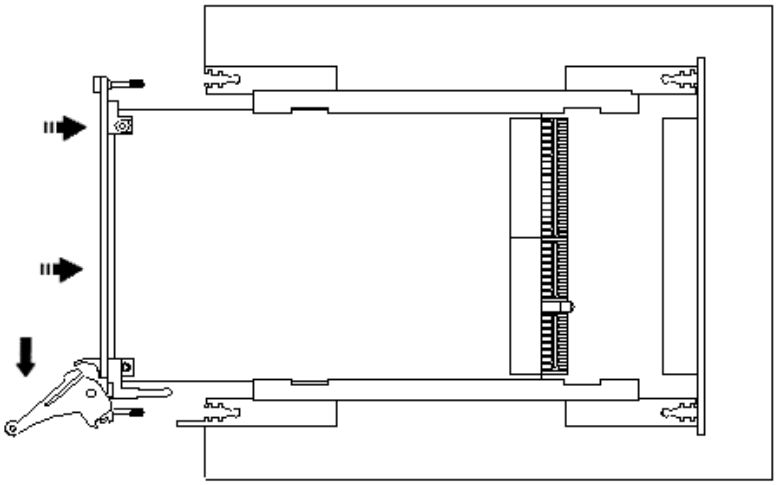

**Figure 3-2: Ejector handles position during module insertion**

5. After you feel resistance, push in the ejector handles as shown in Figure 3-3 to secure the module into the frame.

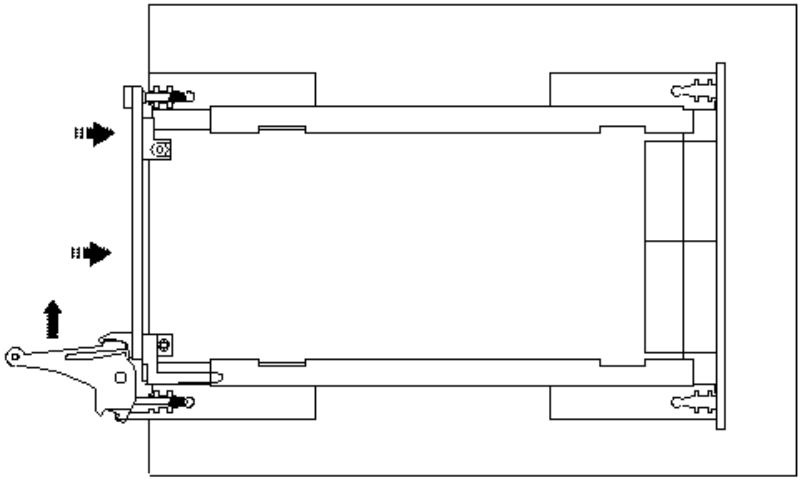

**Figure 3-3: Ejector handles position after module insertion**

6. Tighten the module's front panel to the chassis to secure the module in.
7. Connect any necessary cables to the board.
8. Plug the power cord in and turn on the PXI chassis.

### Plug & Play Driver Installation

Plug & Play operating systems such as Windows notifies the user that a new board was found using the **New Hardware Found** wizard after restarting the system with the new board.

If another Marvin Test Solutions board software package was already installed, Windows will suggest using the driver information file: HW.INF. The file is located in your Program Files\Marvin Test Solutions\HW folder. Click **Next** to confirm and follow the instructions on the screen to complete the driver installation.

If the operating system was unable to find the driver (since the driver was not installed prior to the board installation), you may install the software as described in the prior section, then click on the **Have Disk** button and browse to select the HW.INF file located in C:\Program File\Marvin Test Solutions\HW.

If you are unable to locate the driver click **Cancel** to the found New Hardware wizard and exit the New Hardware Found Wizard, install the GXAO driver, reboot your computer and repeat this procedure.

The Windows Device Manager (open from the System applet from the Windows Control Panel) must display the proper board name before continuing to use the board software (no Yellow warning icon shown next to device). If the device is displayed with an error you can select it and press delete and then press F5 to rescan the system again and to start the New Hardware Found wizard.

### Removing a Board

Remove the board as follows:

1. Turn off the PXI chassis and unplug the power cord.
2. Locate a PXI slot on the PXI chassis.
3. Disconnect and remove any cables/connectors connected to the board.
4. Un-tighten the module's front panel screws to the chassis.
5. Push out the ejector handles and slide the PXI board away from the chassis.
6. Optionally - uninstall the software by running the setup again (or from the Windows Control Pane, Programs and Features or Add Remove Programs applet) and selecting Remove/Uninstall.

## Connectors and Accessories

The following accessories are available from Marvin Test Solutions for your switching board:

| Part / Model Number | Description |
|---|---|
| GT96002 | Connector, D-type 78-pin male with solder pins. |
| GT97102 | 3' harness, 78-pin male connector on one end, loose wires (numbered) other end. |
| GT97103 | 1' harness, 78-pin male connector on one end, loose wires (numbered) other end. |
| GT97104 | 1' harness, 78- pin male connectors on both ends. |
| GT96107 | 3' harness, 78-pin male connector on both ends. |
| GX96106 | 6' harness, 78-pin male connector on both ends. |
| GT96078 | 78-Pin connector to screw terminal interface. |

**Table 3-1: Spare Connectors and Accessories**

## Connectors

Figure 3-4 shows the available GX1648 board connector with description:

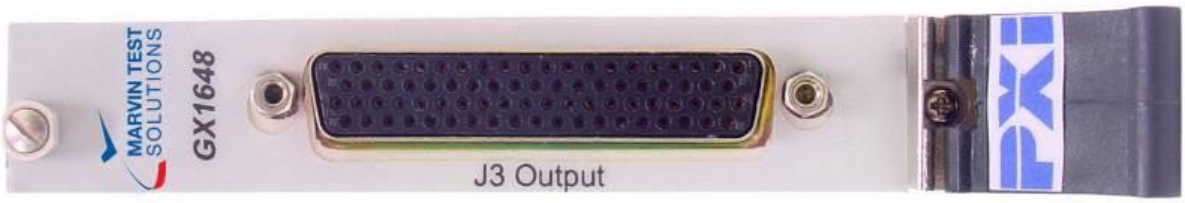


**Figure 3-4: GX1648 J3 Output Channels Connector**

Figure 3-4 shows the available GX1649 board connector with description:

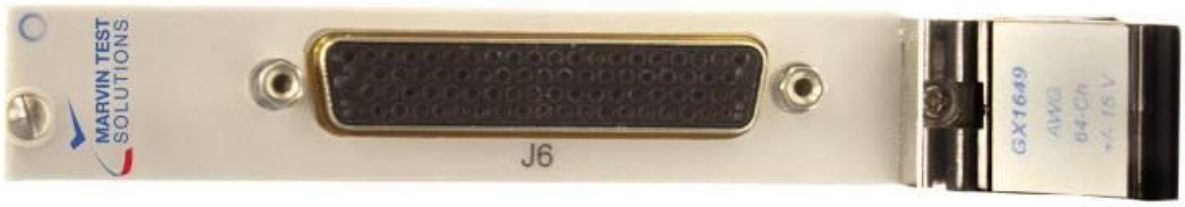


**Figure 3-5: GX1649 J6 Output Channels Connector**

## Output Channels Connector (J3)

The following are the connector signal pin assignments for GX1648/GX1642 module:

| Pin# | Signal | Pin# | Signal | Pin# | Signal | Pin# | Signal |
|---|---|---|---|---|---|---|---|
| 1 | ChA0 | 21 | ChB0 | 40 | ChC0 | 60 | ChD0 |
| 2 | ChA1 | 22 | ChB1 | 41 | ChC1 | 61 | ChD1 |
| 3 | ChA2 | 23 | ChB2 | 42 | ChC2 | 62 | ChD2 |
| 4 | ChA3 | 24 | ChB3 | 43 | ChC3 | 63 | ChD3 |
| 5 | ChA4 | 25 | ChB4 | 44 | ChC4 | 64 | ChD4 |
| 6 | ChA5 | 26 | ChB5 | 45 | ChC5 | 65 | ChD5 |
| 7 | ChA6 | 27 | ChB6 | 46 | ChC6 | 66 | ChD6 |
| 8 | ChA7 | 28 | ChB7 | 47 | ChC7 | 67 | ChD7 |
| 9 | ChA8 | 29 | ChB8 | 48 | ChC8 | 68 | ChD8 |
| 10 | ChA9 | 30 | ChB9 | 49 | ChC9 | 69 | ChD9 |
| 11 | ChA10 | 31 | ChB10 | 50 | ChC10 | 70 | ChD10 |
| 12 | ChA11 | 32 | ChB11 | 51 | ChC11 | 71 | ChD11 |
| 13 | ChA12 | 33 | ChB12 | 52 | ChC12 | 72 | ChD12 |
| 14 | ChA13 | 34 | ChB13 | 53 | ChC13 | 73 | ChD13 |
| 15 | ChA14 | 35 | ChB14 | 54 | ChC14 | 74 | ChD14 |
| 16 | ChA15 | 36 | ChB15 | 55 | ChC15 | 75 | ChD15 |
| 17 | ExUpdateA | 37 | ExUpdateB | 56 | ExUpdateC | 76 | ExUpdateD |
| 18 | External Update All | 38 | GND | 57 | TTL Out | 77 | GND |
| 19 | GND | 39 | GND | 58 | GND | 78 | GND |
| 20 | GND | | | 59 | GND | | |

**Table 3-2: J3 – Output Channels Connector**

Legend

- **ChX0 - ChX15**: Group X (Group A-C, D optional) channel output.
- **ExUpdateA/B/CD**: External update line to the corespending group.
- **External Update All:** External Update of all groups.
- **TTL Out**: Digital TTL out level bit.
- **GND**: Ground.

## Output Channels Connector (J6)

The following are the connector signal pin assignments for GX1649 module:

| Pin# | Signal | Pin# | Signal | Pin# | Signal | Pin | Signal |
|---|---|---|---|---|---|---|---|
| 1 | ChA0 | 21 | ChB0 | 40 | ChC0 | 60 | ChD0 |
| 2 | ChA1 | 22 | ChB1 | 41 | ChC1 | 61 | ChD1 |
| 3 | ChA2 | 23 | ChB2 | 42 | ChC2 | 62 | ChD2 |
| 4 | ChA3 | 24 | ChB3 | 43 | ChC3 | 63 | ChD3 |
| 5 | ChA4 | 25 | ChB4 | 44 | ChC4 | 64 | ChD4 |
| 6 | ChA5 | 26 | ChB5 | 45 | ChC5 | 65 | ChD5 |
| 7 | ChA6 | 27 | ChB6 | 46 | ChC6 | 66 | ChD6 |
| 8 | ChA7 | 28 | ChB7 | 47 | ChC7 | 67 | ChD7 |
| 9 | ChA8 | 29 | ChB8 | 48 | ChC8 | 68 | ChD8 |
| 10 | ChA9 | 30 | ChB9 | 49 | ChC9 | 69 | ChD9 |
| 11 | ChA10 | 31 | ChB10 | 50 | ChC10 | 70 | ChD10 |
| 12 | ChA11 | 32 | ChB11 | 51 | ChC11 | 71 | ChD11 |
| 13 | ChA12 | 33 | ChB12 | 52 | ChC12 | 72 | ChD12 |
| 14 | ChA13 | 34 | ChB13 | 53 | ChC13 | 73 | ChD13 |
| 15 | ChA14 | 35 | ChB14 | 54 | ChC14 | 74 | ChD14 |
| 16 | ChA15 | 36 | ChB15 | 55 | ChC15 | 75 | ChD15 |
| 17 | Ref | 37 | GPIO2 | 56 | Reserve | 76 | GPIO6 |
| 18 | GPIO0 | 38 | GPIO3 | 57 | GPIO4 | 77 | GPIO7 |
| 19 | GPIO1 | 39 | GND | 58 | GPIO5 | 78 | GND |
| 20 | GND | | | 59 | GND | | |

**Table 3-3: J6 – Output Channels Connector**

Legend

- **ChX0 - ChX15**: Group X (Group A-D) channel output.
- 
- **GPIOx**: Digital TTL IO Channels. GPIO0 to GPIO3 serve as the external trigger inputs for GroupA to GroupD ARBs respectively. GPIO0 to GPIO3 also serve as the external update inputs for GroupA to GroupD static outputs.
- **GND**: Ground.

## Installation Directories

The GXAO driver files are installed in the default directory `C:\Program Files\Marvin Test Solutions\GXAO`. You can change the default `GXAO` directory to one of your choosing at the time of installation.

During the installation, `GXAO` Setup creates and copies files to the following directories:

| Name | Purpose / Contents |
|---|---|
| …\Marvin Test Solutions\GXAO | The GXAO software package directory. Contains panel programs, programming libraries, interface files and examples, on-line help files and other documentation. |
| …\Marvin Test Solutions\HW | HW device driver. Provide access to your board hardware resources such as memory, IO ports and PCI board configuration. See the README.TXT located in this directory for more information. |
| …\ATEasy\Drivers | ATEasy drivers directory. GXAO Driver and example are copied to this directory only if ATEasy is installed to your machine. |
| ..\Windows\System32 when running 32 bit Windows or ..\Windows\SysWOW64 for 64 bit Windows | Windows System directory. Contains the GXAO DLL driver and some upgraded system components, such as the HTML help viewer, etc. |

## Driver Files Description

The Setup program copies the GXAO driver, a panel executable, the GXAO help file, the README.TXT file, and driver samples. The following is a brief description of each installation file:

### Driver File and Virtual Panel

- GXAO.DLL and GXAO64.DLL - 32 or 64bit MS-Windows DLL for applications running under Windows, 95, 98, Me, NT, 2000, XP, VISTA, Windows 7.
- GXAOPANEL.EXE and GXAOPANEL64 – An instrument front panel program for all GXAO supported boards for 32 and 64 bit Windows.

### Interface Files

The following GXAO interface files are used to support the various development tools:

- GXAO.H - header file for accessing the DLL functions using the C/C++ programming language. The header file compatible with the following 32-bit development tools:
  - Microsoft Visual C++, Microsoft Visual C++ .NET
  - Borland C++
- GXAO.LIB and GXAO64.LIB - Import library for GXAO DLLs (used when linking C/C++ application that uses GXAO.DLL).
- GXAOBC.LIB - Import library for GXAO.DLL (used when linking Borland C/C++ application that uses GXAO.DLL).
- GXAO.PAS - interface file to support Borland Pascal or Borland Delphi.
- GXAO.BAS - Supports Microsoft Visual Basic 4.0, 5.0 and 6.0.

- GXAO.VB - Supports Microsoft Visual Basic .NET.
- GXAO.LLB - LabView interface for the GXAO DLL.
- Gx1642.drv - ATEasy driver File for Gx1642
- Gx1648.drv - ATEasy driver File for Gx1648
- Gx1649.drv – ATEasy driver file for Gx1649

### On-line Help and Manual

GXAO.CHM – On-line version of the GX1642/GX1648/GX1649 User's Guide. The help file is provided in a Windows Compiled HTML help file (.CHM). The file contains information about the boards, programming reference and panel operation.

GX164xUG.PDF – On line, printable version of the GX1642/GX1648 User's Guide in Adobe Acrobat format. To view or print the file you must have the reader installed. If not, you can download the Adobe Acrobat reader (free) from http://www.adobe.com.

### ReadMe File

README.TXT – Contains important last minute information not available when the manual was printed. This text file covers topics such as a list of files required for installation, additional technical notes, and corrections to the GXAO manuals. You can view and/or print this file using the Windows NOTEPAD.EXE or any other text file editors.

### Example Programs

The sample program includes a C/C++ sample compiled with various development tools, Visual Basic example and an ATEasy sample. Other examples may be available for other programming tools.

#### Microsoft Visual C++ .NET example files:

- GxAoExampleC.cpp - Source file
- GxAoExampleC.ico - Icon file
- GxAoExampleC.rc - Resource file
- GxAoExampleC.vcproj – Project file
- GxAoExampleC.exe - Example executable

#### Microsoft Visual C# .NET Example files:

- GxAoExapmleCS.cs – Source file
- GxAoExampleCS.csproj – Project file

#### Microsoft Visual C++ 6.0 example files:

- GxAoExampleC.cpp - Source file
- GxAoExampleC.ico - Icon file
- GxAoExampleC.rc - Resource file
- GxAoExampleC.dsp - VC++ project file
- GxAoExampleC.exe - Example executable

#### Borland C++ example files:

- GxAoExampleC.cpp - Source file
- GxAoExample.ico - Icon file
- GxAoExampleC.rc - Resource file

- GxAoExampleC.bpr - Borland project file
- GxAoExampleC.exe - Example executable

**Microsoft Visual Basic .NET example files:**

- GxAoExampleVB.vb - Example form.
- GxAoExampleVB.resx - Example form resource.
- GxAoExampleVBapp.config - Example application configuration file.
- GxAoExampleVBAssembleyInfo.vb - Example application assembly file
- GxAoExampleVB.vbproj - Project file
- GxAoExampleVB.exe - Example executable

**Microsoft Visual Basic 6.0 example files:**

- GxAoExampleVB6.frm - Example form
- GxAoExampleVB6.frx - Example form binary file
- GxAoExampleVB6.vbp - Project file
- GxAoExampleVB6.exe - Example executable.

**ATEasy driver and examples files (ATEasy Drivers directory):**

- GxAo.wsp – ATEasy workspace for GX1642.prj,GX1648.prj, and GX1649.prj
- Gx1642.drv - driver
- Gx1642.prj - example project
- Gx1642.sys - example system
- Gx1642.prg - example program
- Gx1648.drv - driver
- Gx1648.prj - example project
- Gx1648.sys - example system
- Gx1648.prg - example program
- Gx1649.drv – driver
- Gx1649.prj – example project
- Gx1649.sys – example system
- Gx1649.prg – example program

## Setup Maintenance Program

You can run Setup again after GXAO has been installed from the original disk or from the Windows Control Panel **Add/Remove Programs** applet. Setup will be in the Maintenance mode when running for the second time. The Maintenance window show below allows you to modify the current GXAO installation. The following options are available in Maintenance mode:

- **Modify.** When you want to add or remove GXAO components.
- **Repair.** When you have corrupted files and need to reinstall.
- **Remove.** When you want to completely remove GXAO.

Select one of the options and click **Next**.

Follow the instruction on the screen until Setup is complete.

# Chapter 4 - Programming the Board

## Overview

This chapter contains information about how to program the switching instruments using the GXAO driver. The GXAO driver contains functions to initialize, reset, and control the switching instruments. A brief description of the functions, as well as how and when to use them, is included in this chapter. Chapter 5 and the specific instrument User's Guide contain a complete and detailed description of the available programming functions.

The GXAO driver supports many development tools. Using these tools with the driver is described in this chapter. In addition, the GXAO directory contains examples written for these development tools. Refer to Chapter 3 for a list of the available examples.

An example using the DLL driver with Microsoft Visual C++ is included at the end of this chapter. Since the driver functions and parameters are identical for all operating systems and development tools, the example can serve as an outline for other programming languages, programming tools, and other GXAO driver types.

## The GXAO Driver

The GXAO driver is a Windows DLL file: GXAO.DLL for 32 bit and GXAO64.DLL for 64 bit Windows. The DLLs are used with 32 bit or 64 bit applications running under Windows. The DLL uses a device driver to access the board resources. The device driver, HW, is installed by the GXAO setup program and is shared by other Marvin Test Solutions products (ATEasy, GXAO, etc).

The DLLs can be used with various development tools such as Microsoft Visual C++, Borland C++ Builder, Microsoft Visual Basic, Borland Pascal or Delphi, ATEasy, LabView and more. The following paragraphs describe how to create an application that uses the driver with various development tools. Refer to the paragraph describing the specific development tool for more information.

## Programming Using C/C++ Tools

The following steps are required to use the GXAO driver with C/C++ development tools:

- Include the GxAo.h header file in the C/C++ source file that uses the GXAO function. This header file is used for all driver types. The file contains function prototypes and constant declarations to be used by the compiler for the application.
- Add the required .LIB file to the projects. This can be import library GXAO.LIB for Microsoft Visual C++ and GXAOBC.LIB for Borland C++. Windows based applications that explicitly load the DLL by calling the Windows **LoadLibrary** API should not include the .LIB file in the project.
- Add code to call the GXAO as required by the application.
- Build the project.
- Run, test, and debug the application.

## Programming Using Visual Basic

To use the driver with Visual Basic 4.0, 5.0 or 6.0 (for 32-bit applications), the user must include the GXAO.BAS to the project. For Visual Basic .NET use the GxAo.vb.

The file can be loaded using *Add File* from the Visual Basic *File menu*. The GxAo.bas contains function declarations for the DLL driver.

## Programming Using Visual Basic.NET

To use the driver with Visual Basic .NET (for 32-bit applications), the user must include the GxAo.vb to the project. For Visual Basic .NET use the GxAo.vb.

The file can be loaded using *Add File* from the Visual Basic *File menu*. The GxAo.vb contains function declarations for the DLL driver.

## Programming Using Pascal/Delphi

To use the driver with Borland Pascal or Delphi, the user must include the GxAo.pas to the project. The GXAO.PAS file contains a **unit** with function prototypes for the DLL functions. Include the GXAO unit in the **uses** statement before making calls to the GXAO functions.

## Programming GXAO Boards Using ATEasy®

The GXAO package is supplied with an ATEasy driver for each of the board types supported by this package. The ATEasy driver uses the GxAo.dll to program the board. Each ATEasy driver includes an example that contains a program and a system file for use with the ATEasy driver. Use the driver shortcut property page from the System Drivers sub-module to correct the board slot number (PCI) or base address (ISA) before attempting to run the example.

Plain language commands declared in the ATEasy driver are easier to use than using the DLL functions directly. The driver commands will also generate exception that allows the ATEasy application to trap errors without checking the status code returned by the DLL function after each function call.

The ATEasy driver commands are similar to the DLL functions in name and parameters, with the following exceptions:

- The *nHandle* parameter is omitted. The driver handles this parameter automatically. ATEasy uses driver logical names instead i.e. AO1 for the first analog output board, AO2 for the second, etc.
- The *nStatus* parameter was omitted. Use the Get Status commands instead of checking the status. After calling a DLL function the ATEasy driver will check the returned status and will call the error statement (in case of an error status) to generate exception that can be easily trapped by the application using the **OnError** module event or using the **try-catch** statement.

Some ATEasy drivers contain additional commands to permit easier access to the board features. For example, parameters for a function may be omitted by using a command item instead of typing the parameter value. The use of plain language commands is self-documenting. The syntax is similar to English. In addition, you may generate the commands from the code editor context menu or by using the ATEasy's code completion feature instead of typing them directly.

## Programming Using LabView and LabView/Real Time

To use the driver with LabView use the provided lab view library GxAo.llb. The library is located in the GXAO folder. An example for LabView is also provided in the Examples folder. A DLL located in the LabViewRT folder can be used for deployment with LabView/Real-Time.

## Using and Programming under Linux

Marvin Test Solutions provides a separate software package with a Linux driver, **GtLinux**. The software package can download from the Marvin Test Solutions website. See the ReadMe.txt in that package for more information regarding using and programming the driver under Linux.

## Using the GXAO driver functions

The GXAO driver contains a set of functions for the GX1642/GX1648 boards. Functions names that starts with the **GxAo** prefix applies to all GXAO boards (i.e. **GxAoReset**). The GXAO functions are designed with consistent set of arguments and functionality. All boards have a function that initializes the GXAO driver for a specific board, reset the board, and display the virtual panel. All the functions use handles to identify and reference a specific board and all functions return status and share the same functions to handle error codes.

### Initialization, HW Slot Numbers and VISA Resource

The GXAO driver supports two device drivers HW and VISA which are used to initialize, identify and control the board. The user can use the **GxAoInitialize** to initialize the board 's driver using HW and **GxAoInitializeVisa** to initialize using VISA. The following describes the two different methods used:

1. **Marvin Test Solutions's HW** - the default device driver that is installed by the GXAO driver. To initialize and control the board using the HW use the **GxAoInitialize**(*nSlot, pnHandle, pnStatus*) function. The function initializes the driver for the board at the specified PXI slot number (*nSlot*) and returns a board handle. The **PXI/PCI Explorer** applet in the Windows Control Panel displays the PXI slot assignments. You can specify the *nSlot* parameter in the following way:

   - A combination of chassis number (chassis # x 256) with the chassis slot number, e.g. 0x105 for chassis 1 and slot 5. Chassis number can be set by the **PXI/PCI Explorer** applet.
   - Legacy nSlot as used by earlier versions of HW/VISA. The slot number contains no chassis number and can be changed using the **PXI/PCI Explorer** applet: 23 in this example.

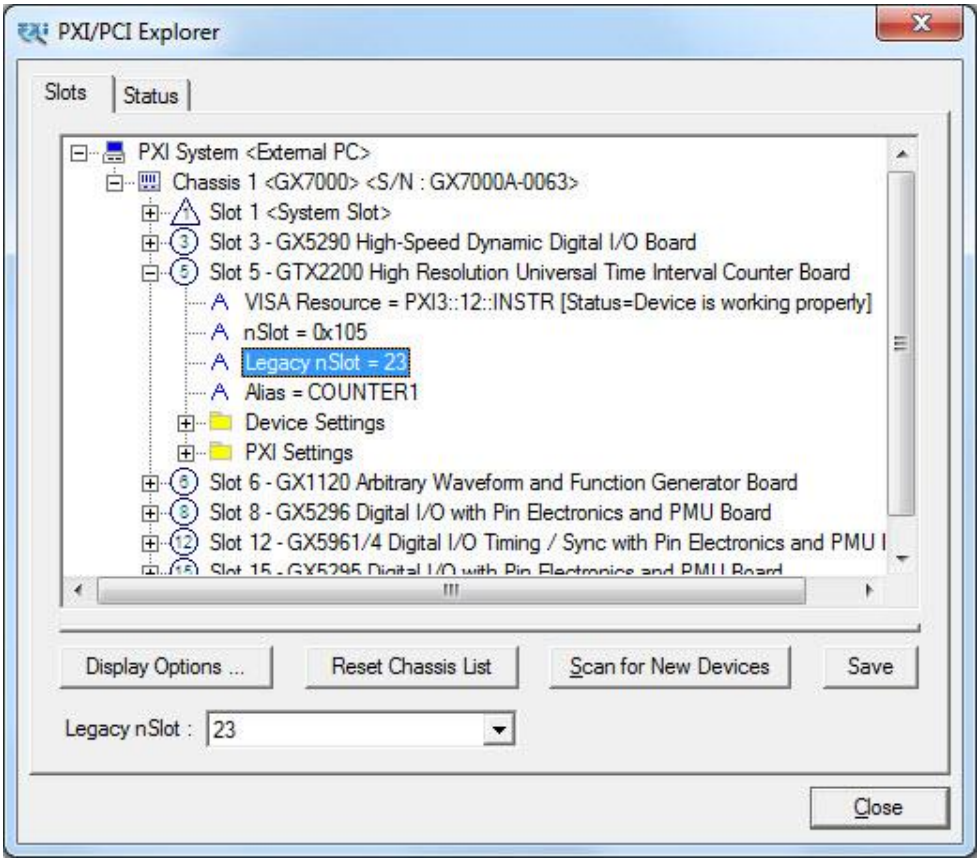


**Figure 4-1: PXI/PCI Explorer**

2. **VISA** – this is a third party library usually by National Instruments (NI-VISA). You must ensure that the VISA installed supports PXI and PCI devices (not all VISA providers supports PXI/PCI). GXAO setup installs a VISA compatible driver for the GXAO board in-order to be recognized by the VISA provider. Use the GXAO function **GxAoInitializeVisa** (*szVisaResource, pnHandle, pnStatus*) to initialize the driver board using VISA. The first argument *szVisaResource* is a string that is displayed by the VISA resource manager such as NI **Measurement and Automation** (NI_MAX). It is also displayed by Marvin Test Solutions **PXI/PCI Explorer** as shown in the prior figure. The VISA resource string can be specified in several ways as the following examples:

    - Using chassis, slot: "PXI0::CHASSIS1::SLOT5"
    - Using the PCI Bus/Device combination: "PXI9::13::INSTR" (bus 9, device 9).
    - Using alias: "COUNTER1". Use the PXI/PCI Explorer to set the device alias.

    Information about VISA is available at http://www.pxisa.org.

The **GxAoInitialize** function returns a handle that is required with other driver functions to program the board. This handle is usually saved in the program in a global variable for later use when calling other functions. The initialize function does not change the state of the board or its settings.

### Board Handle

The board handle argument, *nHandle* , passed (by reference) to the parameter *pnHandle* of the **GxAoInitialize** or the **GxAoInitializeVisa** functions is a short integer (16 bits) number. It is used by the GXAO driver functions to identify the board being accessed by the application. Since the driver supports many boards at the same time, the *nHandle* argument is required to uniquely identify which board is being programmed.

The *nHandle* is created when the application calls the **GxAoInitialize** function. But there is no need to destroy the handle. Calling **GxAoInitialize** with the same slot number will return the same handle.

Once the board is initialized the handle can be used with other functions to program the board.

### Reset

The Reset function causes the driver to change all settings to their default state. The application software issue a Reset after the initializing the Counter, but a Reset can be issued any time. All counter boards have the **GxAoReset**(*nHandle, nStatus*) function. See the Function Reference for more information regarding the specific board.

### Error Handling

All GXAO functions pass a fail or success status - *pnStatus* - in the last parameter. A successful function call passes zero in the status parameter upon return. If the status is non-zero, then the function call fails. This parameter can be later used for error handling. When the status is error, the program can call the **GxAoGetErrorString** function to return a string representing the error. The **GxAoGetErrorString** reference contains possible error numbers and their associated error strings.

### Driver Version

The **GxAoGetDriverSummary** function can be used to return the current GXAO driver version. It can be used to differentiate between the driver versions. See the Function Reference for more information.

### Panel

Calling the **GxAoPanel** will display the instrument's front panel dialog window. The panel can be used to initialize and control the board interactively. The panel function may be used by the application to allow the user to directly interact with the board.

The **GxAoPanel** function is also used by the GXAOPANEL.EXE or GXAOPANEL64.EXE panel programs that is supplied with this package and provides a stand-alone Windows application that displays the instrument panel.

## Distributing the Driver

Once the application is developed, the driver files (GxAo.dll or GxAo 64.dll and the HW device driver files located in the HW folder) can be shipped with the application. Typically, the DLLs should be copied to the Windows System directory. The HW device driver files should be installed using a special setup program HWSETUP.EXE that is provided with GXAO driver files. Alternatively, you can provide the GxAo.exe setup to be installed along with the board.

## Sample Programs

The following example demonstrates how to program the board using the C programming language under Windows. The example shows how to get or set a group or channel voltage.

To run, enter the following command line:

**GxAoExample** <Slot> <operation> <group> <channel> <voltage>

Where:

| | |
|---|---|
| <Slot> | Board PXI Slot number where the board resides. |
| <Operation> | Operation code:<br>GCV=Get channel voltage<br>SCV=Set channel voltage<br>SVR=Set group voltage range<br>GVR=Get group voltage range |
| <Group > | Group number: 0 to 2 |
| <Channel\|Range> | Channel number: 0 to 15<br>Voltage Range: 0 to 3 |
| <Value> | Set or get value. |

## Sample Program Listing

```
/*****************************************************************************
      FILE        : GxAoExampleC.cpp

      PURPOSE           : WIN32/Linux sample program for GX1642/GX1648 boards
                          using the GXAO driver.

      CREATED           : Aug. 2002

      COPYRIGHT   : Copyright 2002-2010 MARVIN TEST SOLUTIONS.

      COMMENTS    :

      To compile the example:

      1. Microsoft VC++
            Load GxAoExampleC.dsp, .vcproj or .mak, depends on
            the VC++ version from the Project\File/Open... menu
            Select Project/Rebuild all from the menu

      2. Borland C++ Builder
            Load GxAoExampleC.bpr from the Project/Open
                  Project... menu
            Select Project/Build all from the menu

      3. Linux (GCC for CPP and Make must be available)
            make -fGxAoExampleC.mk [CFG=Release[64] | Debug[64]] [rebuild |
                clean]

*****************************************************************************/
#ifndef __GNUC__
#include "windows.h"
#endif
#include "GxAo.h"
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <ctype.h>

#if defined(__BORLANDC__)
#pragma hdrstop
#include <condefs.h>
USELIB("GxAoBC.lib");
USERC("GxAoExampleC.rc");
#endif

//*****************************************************************************
//          DisplayMsg
//*****************************************************************************
void DisplayMsg(PCSTR lpszMsg)
{
#ifndef __GNUC__
      MessageBeep(0);
      MessageBox(0, lpszMsg, "GxAo example program", MB_OK);
```

```
#else
    printf("\r\nGxAo example program: %s\r\n", lpszMsg);
#endif
      return;
}

//*************************************************************************
//           __strupr
//*************************************************************************
char * __strupr(char * sz)
{
    int i;

    for (i=0; sz[i]; i++)
        sz[i] = toupper(sz[i]);
    return sz;
}

//*************************************************************************
//          DisplayUsage
//*************************************************************************
void DisplayUsage(void)
{
      DisplayMsg(
            "\r\nThis example shows how to use the GX1642/GX1648/GX1649:"
            "\r\n\r\n""

            "Usage:\r\n"
            "GxAoExample <slot|address> <operation> <group> <channel|range>"
            " <voltage>"

            "\r\n\r\nWhere : \r\n"
        "<slot> - Under Windows: PCI/PXI slot number as shown by the PXI"
        " explorer\r\n"
        "<address> - Under Linux: Bus/device as displayed with lspci"
        " utility\r\n"

        "<operation> - one of the followings :\r\n"
            "\tSCV=Set channel voltage\r\n"
            "\tGCV=Get channel value\r\n"
            "\tSVR=Set group voltage range\r\n"
            "\tGVR=Get group voltage range\r\n"
            "\tSUM=Print board summary\r\n"
            "<group> - group number 0 to 3\r\n"
            "<channel|range> - depends on the operation:\r\n"
            "\tchannel number :\t0-15 for SCV/GCV operations\r\n"
            "\trange number :\t0-3 for SVR/GVR operations\r\n"
            "<voltage> - channel's voltage\r\n"

            "\r\nThe example should be run from  the Windows command"
            " prompt/Linux terminal.\r\n"
        "\tExamples:\r\n"
        "\t./GxAoExampleC 7 SCV 0 1 2.5\r\n (group 0 chan 1, 2.5v)"
        "\t./GxAoExampleC 0x60c SCV 0 1 2.5 (under Linux, bus 6 device 12)"
        "\r\n"
            );
      exit(1);
```

```
}

//*************************************************************************
//          CheckStatus
//*************************************************************************
void CheckStatus(SHORT nStatus)
{
    CHAR    sz[128];

      if (!nStatus) return;
      GxAoGetErrorString(nStatus, sz, sizeof sz, &nStatus);
      DisplayMsg(sz);
      DisplayMsg("Aborting the program...");
      exit(nStatus);
}

//*************************************************************************
//          MAIN
//
// This main function receives five parameters
//
// GX164x board slot number (e.g. 1)
// GX164x operation (e.g. GCV=get channel voltage)
//          SCV=set channel voltage,
//          GCV=get channel voltage
//          SVR=set group voltage range
//          GVR=get group voltage range
//      SUM=print board summary
// GX164x channel/voltage_range depends on operation
//                      0 - 15 for channel
//                      0 - 3 for voltage_range
// GX164x voltage value double
//
//*************************************************************************
int main(int argc, char **argv)
{
      short nSlotNum;         // Board slot number
      char *      sOperation;       // Board Operation
      short nGroup=0;               // Group number
      short nChannel_Range;   // Channel or Voltage Range
      short nHandle;          // Board handle
      short nStatus;          // Returned status
      double      dVoltage;         // Channel's voltage
      char    sz[512];        // board summary

      // Check number of arguments recived
      if (argc<3) DisplayUsage();

      // Parse command line parameters
      nSlotNum=(SHORT)strtol(*(++argv), NULL, 0);
      sOperation = __strupr(*(++argv));
      if (strcmp(sOperation, "SUM"))
      {   nGroup=(SHORT)strtol(*(++argv), NULL, 0);
        nChannel_Range =(SHORT)strtol(*(++argv), NULL, 0);
      }
      if (nSlotNum<0 || nGroup<0 || nGroup>3)
            DisplayUsage();
```

```
        GxAoInitialize(nSlotNum, &nHandle, &nStatus);
        CheckStatus(nStatus);

        if (!strcmp(sOperation, "SCV"))
        {      // set channel voltage
               if (argc<5) DisplayUsage();
               dVoltage=strtod(*(++argv), NULL);
               GxAoSetChannelVoltage(nHandle, nGroup, nChannel_Range, dVoltage,
               &nStatus);
               CheckStatus(nStatus);
               printf("Set Group %i Channel %i voltage to %f.\n", nGroup,
               nChannel_Range, dVoltage);
        }
        else if (!strcmp(sOperation, "GCV"))
        {      // get channel voltage
               GxAoGetChannelVoltage(nHandle, nGroup, nChannel_Range, &dVoltage,
               &nStatus);
               CheckStatus(nStatus);
               printf("Group %i Channel %i voltage is %f.\n", nGroup,
               nChannel_Range, dVoltage);
        }
        else if (!strcmp(sOperation, "SVR"))
        {      // set group voltage range
               GxAoSetGroupVoltageRange(nHandle, nGroup, nChannel_Range,
               &nStatus);
               CheckStatus(nStatus);
               printf("Group %i voltage range to %i.\n", nGroup,
               nChannel_Range);
        }
        else if (!strcmp(sOperation, "GVR"))
        {      // set group voltage range
               GxAoGetGroupVoltageRange(nHandle, nGroup, &nChannel_Range,
               &nStatus);
               CheckStatus(nStatus);
               printf("Group %i voltage range is %i.\n", nGroup,
               nChannel_Range);
        }
        else if (!strcmp(sOperation, "SUM"))
      {   // print board summary
          GxAoGetBoardSummary(nHandle, sz, sizeof sz, &nStatus);
               CheckStatus(nStatus);
               printf("Board Summary: %s.\n", sz);
      }

        return 0;
}

//*************************************************************************
//          End Of File
//*************************************************************************
```

# Chapter 5 - Calibration

## Introduction

The GX1649 can be calibrated using the GXAO driver API or by using Marvin Test Solutions's **CalEasy** software. This chapter focuses on using the driver API for calibration. Using either method of calibration requires purchasing of a calibration license for CalEasy or for the API from Marvin Test Solutions.

The GXAO driver API exposes a set of functions to allow the end user to create their own calibration program. The GX1649 stores all calibration data within an onboard EEPROM.

## Hardware Requirements

In order to calibrate the GX1649 the user must have access to a high accuracy 8 ½ DMM. The DMM must be able to measure +/- 15 Volts.

## Calibration Licensing

A calibration license must be obtained from Marvin Test Solutions in order to unlock the Calibration functionality.

The software front panel is used to enter a valid license string using the following procedure:

1. Initialize the software front panel:

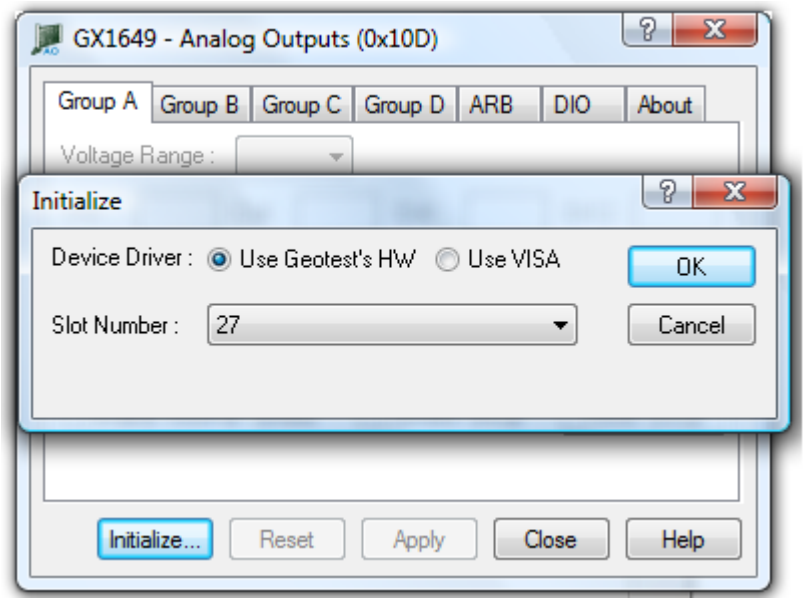


**Figure 5-1: Gx1642/8/9 – Initialize dialog**

2. Click on the About Tab:

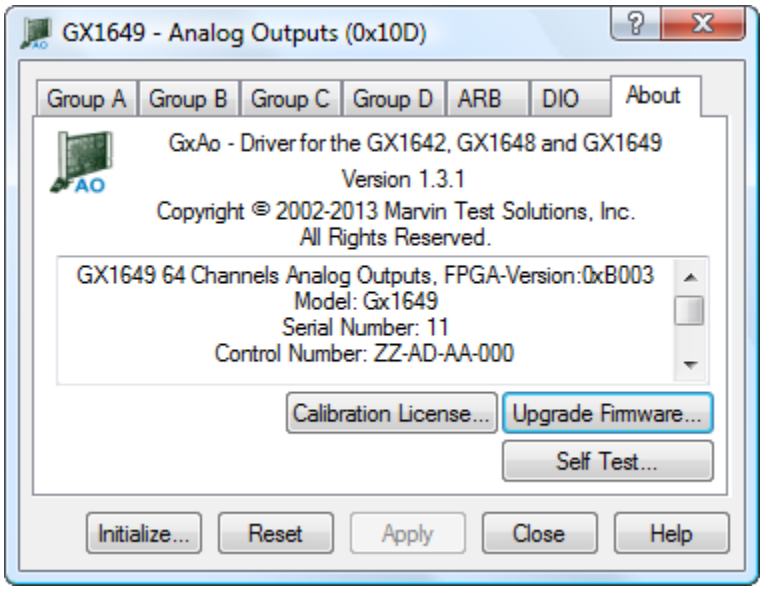


**Figure 5-2: GX1649 – About page**

3. Click on the Calibration License… button.
4. Make note of the Computer ID. Send this ID to Marvin Test Solutions in exchange for a License String.
5. Fill in the appropriate fields including the License String and click OK.

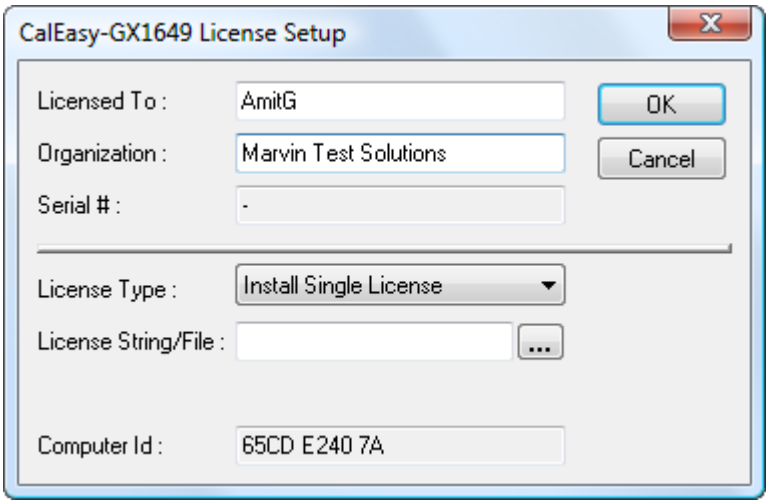


**Figure 5-3: CalEasy - GX1649 License Setup**

At this point the Calibration functionality has been unlocked on a specific computer.

## GXAO Functions used for Calibration

The functions used for calibration are:

| Function | Description |
|---|---|
| **GxAoInitialize** | Initializes the GX1649 |
| **GxAoCalSetMode** | Set the calibration mode (enabled or disabled). Used to start the calibration procedure |
| **GxAoCalSetVoltagePoint** | Set voltage point for ADC calibration |
| **GxAoCalADCMeasure** | Measure ADC at voltage points |
| **GxAoCalADC** | Set ADC calibration measurements |
| **GxAoCalGroupChannel** | Calibration a channel (within a group) using the onboard ADC (after it has been calibrated using **GxAoCalADC**) |
| **GxAoCalWriteEEPROM** | Finalize calibration by writing calibration constants to the onboard EEPROM |

**Table 5-1: GXAO Calibration Functions**

For further information on these functions, review the **Function Reference** section of this manual.

Use the CalEasy user guide for connection information.

## Calibration Procedure

### Initial Calibration Procedure

1. Initialize the GX1649 (**GxAoInitialize**)
2. Reset GX1649
3. Enable calibration mode (**GxAoCalSetMode**)

### ADC Voltage Calibration

1. Connect DMM positive terminal to GX1649 J6 pin 1 (Group A, Channel 0) and connect DMM negative terminal to GX1649 J6 pin 20 (Ground)
2. Set calibration voltage point to negative on Group A Channel 0 (**GxAoCalSetVoltagePoint)**
3. Take a Measurement from the reference DMM
4. Take a Measurement from the onboard ADC (**GxAoCalADCMeasure**)
5. Set calibration voltage point to positive on Group A Channel 0 (**GxAoCalSetVoltagePoint**)
6. Take a Measurement from the reference DMM
7. Take a Measurement from the onboard ADC (**GxAoCalADCMesasure**)
8. Calibrate ADC by passing all four measurements from above to the API (**GxAoCalADC)**

### Channel Voltage Calibration

1. Connect J6 pin that corresponds to the channel to be calibrated and J6 pin 20 (Ground) to the external DMM.
2. Set calibration voltage point to negative for the group and channel to be calibrated
3. Measure channel using the external DMM

4. Set calibration voltage point to positive for the group and channel to be calibrated
5. Measure channel using the external DMM
6. Repeat steps 1 to 5 for all channels
7. Calibrate the channel by passing the two measurements from above to the API(**GxAoCalGroupChannel**)

### Finalize Calibration

1. Write calibration to the onboard EEPROM (**GxAoCalWriteEEPROM**)

# Chapter 6 - Functions Reference

## Introduction

The GXAO driver functions reference chapter is organized in alphabetical order. Each function description contains the function name; purpose, syntax, parameters description and type followed by Comments, an Example (written in C), and a See Also sections.

All function parameters syntax follows the same rules:

- Strings are ASCIIZ (null or zero character terminated).
- The first parameter of most functions is *nHandle* (16-bit integer). This parameter is required for operating the board and is returned by the **GxAoInitialize** function. The *nHandle* is used to identify the board when calling a function for programming and controlling the operation of that board.
- All functions return a status with the last parameter named *pnStatus*. The *pnStatus* is zero if the function was successful, or less than a zero on error. The description of the error is available using the **GxAoGetErrorString** function or by using a predefined constant, defined in the driver interface files: GXAO.H, GXAO.BAS, GXAO.VB, GXAO.PAS or GX1642.DRV, GX1648.DRV or GXAO.LLB.
- Group numbers are specified as *nGroup* with 0 to 3 for groups A to D. However, you may only have group A to C installed.

- Parameter name are prefixed as follows:

| Prefix | Type | Example |
|---|---|---|
| *a* | Array - prefix this before the simple type. | *anArray* (Array of Short) |
| *b* | BOOL – Boolean, 0 for FALSE; <>0 for TRUE | *bUpdate* |
| *d* | DOUBLE - 8 bytes floating point | *dReading* |
| *dw* | DWORD - double word (unsigned 32-bit) | *dwTimeout* |
| *hwnd* | Window handle (32-bit integer). | *hwndPanel* |
| *l* | LONG - (signed 32-bit) | *lBits* |
| *n* | SHORT - (signed 16-bit) | *nMode* |
| *p* | Pointer - Usually used to return a value. Prefix this before the simple type. | *pnStatus* |
| *sz* | Null - (zero value character) terminated string | *szMsg* |
| *uc* | BYTE - (8 bits) unsigned. | *ucValue* |
| *w* | WORD - Unsigned short (unsigned 16-bit) | *wParam* |

**Table 6-1: Parameter Name Prefixes**

## GXAO Functions

The following list is a summary of functions available for the GXAO:

| Driver Functions | Description |
|---|---|
| **General** | |
| **GxAoInitialize** | Initializes the driver for the specified slot using the HW device driver. |
| **GxAoInitializeVisa** | Initializes the driver for the specified slot using VISA. |
| **GxAoPanel** | Opens a virtual panel used to interactively control the GXAO. |
| **GxAoReset** | Resets the GXAO board to its default settings. |
| **GxAoGetBoardSummary** | Returns the board summary from the on-board EEPROM. |
| **GxAoGetCalibrationInfo** | Returns the calibration information. |
| **GxAoGetDriverSummary** | Returns the driver name and version. |
| **GxAoGetErrorString** | Returns the error string associated with the specified error number. |
| **Analog Functions** | |
| **GxAoGetBoardAccuracy** | Returns the board minimum per channel accuracy. |
| **GxAoGetChannelVoltage** | Returns the specified channel's group voltage. |
| **GxAoGetGroupExternalUpdate** | Returns the specified group external update enable state. |
| **GxAoGetGroupUpdateState** | Returns whether the group channels loaded values were loaded to it DACs. |
| **GxAoLoadChannelVoltage** | Loads the specified groups' channel with new voltage value. |
| **GxAoResetGroup** | Reset the specified group to the default state. |
| **GxAoSelfTest** | Tests each channel for static voltage accuracy (GX1649). |
| **GxAoSetGroupExternalUpdate** | Enables the specified group external update. |
| **GxAoSetChannelVoltage** | Sets the specified channel voltage. |
| **GxAoUpdateAllGroupsVoltage** | Update all groups' channels outputs with the latest load values. |
| **GxAoUpdateGroupVoltage** | Update all the specified group channels outputs with the load latest values. |
| **Digital I/O Functions (GX1642/GX1648)** | |
| **GxAoGetPortBit** | Returns the digital output bit state. |
| **GxAoSetPortBit** | Sets the digital output bit state. |
| **Voltage Range Functions (GX1642/GX1648)** | |
| **GxAoGetGroupVoltageRange** | Return the specified group voltage range. |
| **GxAoSetGroupVoltageRange** | Sets the specified group voltage range. |

| **Arbitrary Waveform Functions (GX1649 and GX1649-1)** | |
|---|---|
| **GxAoArbArmGroup** | Arms the group. |
| **GxAoArbDisableStreamingInterrupt** | Disables the ARB streaming interrupt. |
| **GxAoArbEnableGroupStreaming** | Enables or Disables streaming mode for a particular ARB group. |
| **GxAoArbGetGroupChannels** | Returns the group channels used for arbitrary waveform generation and the waveform size. |
| **GxAoArbGetGroupClock** | Returns the group clock source and frequency. |
| **GxAoArbGetGroupStatus** | Returns the group's ARB status. |
| **GxAoArbGetGroupStreamingStatus** | Returns the group's ARB streaming status. |
| **GxAoArbGetGroupTrigger** | Returns the group trigger mode. |
| **GxAoArbIsGroupStreaming** | Returns the streaming state for a particular ARB group. |
| **GxAoArbReadChannelWaveform** | Returns the waveform for the specified channel. |
| **GxAoArbResumeStreamingInterrupt** | Resume ARB streaming interrupt handling after processing an interrupt event. |
| **GxAoArbSetGroupChannels** | Sets the group channels used for arbitrary waveform generation and the waveform size. |
| **GxAoArbSetGroupClock** | Sets the group clock source and frequency. |
| **GxAoArbSetGroupTrigger** | Sets the group trigger mode. |
| **GxAoArbSetupStreamingInterrupt** | Sets up the streaming interrupt. Selects which ARB groups are participating in the streaming mode and configures the streaming interrupt call back function. |
| **GxAoArbTrigGroup** | Triggers and starts waveform generation. |
| **GxAoArbWriteChannelWaveform** | Write the channel waveform when in non streaming mode. |
| **GxAoArbWriteStreamingData** | Write to the streaming FIFO when in streaming mode. |
| **Dynamic Digital I/O Functions (GX1649)** | |
| **GxAoDioArm** | Arm dynamic Digital IO sequencer. |
| **GxAoDioGetClock** | Returns the Digital IO Sequencer clock source and frequency. |
| **GxAoDioGetOutputEnable** | Returns the Digital IO output enable. |
| **GxAoDioGetStatus** | Returns the Digital IO sequence's status. |
| **GxAoDioGetVectorCount** | Returns the number of digital vectors (steps) to run |
| **GxAoDioReadMemory** | Read dynamic direction (tri-state control) memory, input or output memory for all 8 DIO channels. |
| **GxAoDioReadChannelMemory** | Read dynamic input (record) memory, input or output memory for a particular DIO channel. |
| **GxAoDioSetClock** | Sets the Digital IO Sequencer clock source and frequency |
| **GxAoDioSetOutputEnable** | Sets the Digital IO output enable. |
| **GxAoDioSetVectorCount** | Sets the number of digital vectors to run. |

| | |
|---|---|
| **GxAoDioTrig** | Triggers and starts Digital IO generation. |
| **GxAoDioWriteMemory** | Write to dynamic direction (tri-state control), input, or output memory for all 8 DIO channels. |
| **GxAoDioWriteChannelMemory** | Write to dynamic input (record) memory, input, or output memory for a particular DIO channel. |
| **Calibration Functions (GX1649)** | |
| **GxAoCalSetVoltagePoint** | Set calibration voltage point for the ADC and Channel. |
| **GxAoCalADCMeasure** | Measure a channel's voltage using onboard Calibration ADC. |
| **GxAoCalADC** | Set the calibration measurements for the ADC. |
| **GxAoCalGroupChannel** | Calibrate a channel by using an external DMM. |
| **GxAoCalSetMode** | Set the calibration mode (enabled or disabled). Used to start the calibration procedure. |
| **GxAoCalWriteEEPROM** | Finalize calibration by writing to the onboard EEPROM. |

## GxAoArbArmGroup

### Purpose

Arms or disables arming the specified group.

### Applies

**GX1649**

### Syntax

**GxAoArbArmGroup** (*nHandle, nGroup, bArm, bContinuous, pnStatus*)

### Parameters

| Name | Type | Comments |
|---|---|---|
| *nHandle* | SHORT | Handle to a GX1649 board. |
| *nGroup* | SHORT | Group number:<br>0. GXAO_GROUPA<br>1. GXAO_GROUPB<br>2. GXAO_GROUPC<br>3. GXAO_GROUPD |
| *bArm* | BOOL | TRUE (1) to arm the group and FALSE (0) to disable arming and stop generating waveforms. |
| *bContinuous* | BOOL | TRUE (1) will cause the waveform to be repeated (until *bArm* is FALSE). FALSE (0) will cause only one waveform to be generated. |
| *pnStatus* | PSHORT | Returned status: 0 on success, negative number on failure. |

### Comments

Disabling arming the group (*bArm*=FALSE) will cause the card to stop generating waveforms.

### Example

The following example generates a repeated waveform for 1000 mSec in group A:

```
SHORT nStatus;
SHORT nTriggerMode;

GxAoArbArmGroup (nHandle, GXAO_GROUPA, TRUE, TRUE, &nStatus);
Sleep(1000)
GxAoArbArmGroup (nHandle, GXAO_GROUPA, FALSE, TRUE, &nStatus);
```

### See Also

**GxAoArbTrigGroup, GxAoArbSetGroupTrigger, GxAoGetErrorString**

## GxAoArbDisableStreamingInterrupt

### Purpose

Disables ARB streaming interrupt.

### Applies

**GX1649-1**

### Syntax

**GxAoArbDisableStreamingInterrupt** (*nHandle*, *pnStatus*)

### Parameters

| Name | Type | Comments |
|---|---|---|
| *nHandle* | SHORT | Handle to a GX1649-1 board. |
| *pnStatus* | PSHORT | Returned status: 0 on success, negative number on failure. |

### Comments

This function disables the ARB streaming interrupt. The interrupt is unregistered by the operating system when this function is called.

### Example

The following example disables the ARB streaming interrupt:

```
SHORT nStatus;

GxAoArbDisableStreamingInterrupt (nHandle, nStatus);
```

### See Also

**GxAoArbEnableGroupStreaming, GxAoArbSetupStreamingInterrupt, GxAoArbResumeStreamingInterrupt, GxAoGetErrorString**

## GxAoArbEnableGroupStreaming

### Purpose

Enables the group streaming mode.

### Applies

**GX1649-1**

### Syntax

**GxAoArbEnableGroupStreaming** (*nHandle, nGroup, bEnable, pnStatus*)

### Parameters

| Name | Type | Comments |
|---|---|---|
| *nHandle* | SHORT | Handle to a GX1649-1 board. |
| *nGroup* | SHORT | Group number:<br>0. GXAO_GROUPA<br>1. GXAO_GROUPB<br>2. GXAO_GROUPC<br>3. GXAO_GROUPD |
| *bEnable* | BOOL | Enables a group's ARB streaming state. TRUE if ARB Group is to be set to streaming mode and FALSE if ARB group is set to non-streaming mode. |
| *pnStatus* | PSHORT | Returned status: 0 on success, negative number on failure. |

### Comments

The GX1649 ARB can operate in streaming or non-streaming mode. By default, each group (Group A-D) is set to non-streaming mode. When the board is in non-streaming mode, the ARB will use the onboard memory as a source for the waveform it will generate. When the board is in streaming mode, the ARB will continuously generate a waveform. The PC will write samples to a FIFO located on the ARB as needed to maintain the streaming operation.

### Example

The following example enables group B streaming mode:

```
SHORT nStatus;
BOOL  bEnable;
GxAoArbEnableGroupStreaming (nHandle, GXAO_GROUPB, TRUE, &nStatus);
```

### See Also

**GxAoArbDisableStreamingInterrupt, GxAoArbGetGroupStreamingStatus, GxAoArbSetupStreamingInterrupt, GxAoGetErrorString**

## GxAoArbGetGroupChannels

### Purpose

Returns the group channels used for arbitrary waveform generation and the waveform size.

### Applies

**GX1649**

### Syntax

**GxAoArbGetGroupChannels** (*nHandle, nGroup, pwChannels, pdwWaveformSize, pnStatus*)

### Parameters

| Name | Type | Comments |
|---|---|---|
| *nHandle* | SHORT | Handle to a GX1649 board. |
| *nGroup* | SHORT | Group number:<br>0. GXAO_GROUPA<br>1. GXAO_GROUPB<br>2. GXAO_GROUPC<br>3. GXAO_GROUPD |
| *pwChannels* | PWORD | Returned a bit mask where each bit corresponds to a channel (bit 0 is channel 0, bit 15 channel 15). Bits that are set to 1 indicate that the corresponding channels are set to generate waveform. Bits that are set to 0 indicate a static output channel. |
| *pdwWaveformSize* | PDWORD | Returned waveform size. |
| *pnStatus* | PSHORT | Returned status: 0 on success, negative number on failure. |

### Comments

All channels configured to generate waveform have the size waveform length.

### Example

The following example checks if channel 2 in group B is static

```
SHORT nStatus;
WORD  wChannels;
DWORD dwWaveformLength;

GxAoArbGetGroupChannels(nHandle, GXAO_GROUPB, &wChannels, &dwLength, &nStatus);
if (wChannels & 0x4)=-0) printf("Channel 2 is static");
```

### See Also

**GxAoArbSetGroupChannels, GxAoArbReadChannelWaveform, GxAoGetErrorString**

## GxAoArbGetGroupClock

### Purpose

Returns the specified group clock source and frequency.

### Applies

**GX1649**

### Syntax

**GxAoArbGetGroupClock** (*nHandle, nGroup, pnClockSource, pdFrequency, pnStatus*)

### Parameters

| Name | Type | Comments |
|---|---|---|
| *nHandle* | SHORT | Handle to a GX1649 board. |
| *nGroup* | SHORT | Group number:<br>0. GXAO_GROUPA<br>1. GXAO_GROUPB<br>2. GXAO_GROUPC<br>3. GXAO_GROUPD |
| *pnClockSource* | PSHORT | Returned clock source:<br>0. GXAO_1649_CLOCKSOURCE_INTERNAL<br>1. GXAO_1649_CLOCKSOURCE_EXTERNAL<br>2. GXAO_1649_CLOCKSOURCE_PXI0<br>3. GXAO_1649_CLOCKSOURCE_PXI1<br>4. GXAO_1649_CLOCKSOURCE_PXI2<br>5. GXAO_1649_CLOCKSOURCE_PXI3<br>6. GXAO_1649_CLOCKSOURCE_PXI4<br>7. GXAO_1649_CLOCKSOURCE_PXI5<br>8. GXAO_1649_CLOCKSOURCE_PXI6<br>9. GXAO_1649_CLOCKSOURCE_PXI7<br>10. GXAO_1649_CLOCKSOURCE_STAR<br>11. GXAO_1649_CLOCKSOURCE_GROUPA |
| *pdFrequncy* | PDOUBLE | Returned group frequency, this is only relevant when pnClockSource is control by the card internally (GXAO_1649_CLOCKSOURCE_INTERNAL) |
| *pnStatus* | PSHORT | Returned status: 0 on success, negative number on failure. |

### Comments

Default clock source is GXAO_1649_CLOCKSOURCE_INTERNAL with frequency of 10MHz GXAO_1649_CLOCKSOURCE_GROUPA can only be used when *nGroup* is set to B, C, and D, using group A clock source will synchronizes all group channels to use the same clock source as group A.

### Example

The following example returns the clock source and frequency for group A:

```
SHORT nStatus;
SHORT nClockSource;
DOUBLE dFrequncy;

GxAoArbGetGroupClock (nHandle, GXAO_GROUPA, &nClockSource, &dFrequency, &nStatus);
```

### See Also

**GxAoArbSetGroupClock, GxAoGetErrorString**

## GxAoArbGetGroupStatus

### Purpose

Returns the group ARB sequencer status.

### Applies

**GX1649**

### Syntax

**GxAoArbGetGroupStatus** (*nHandle, nGroup, pdwStatus, pnStatus*)

### Parameters

| Name | Type | Comments |
|---|---|---|
| *nHandle* | SHORT | Handle to a GX1649 board. |
| *nGroup* | SHORT | Group number:<br>0. GXAO_GROUPA<br>1. GXAO_GROUPB<br>2. GXAO_GROUPC<br>3. GXAO_GROUPD |
| *pdwStatus* | PDWORD | Returned the status of a Group's ARB:<br>Bit 0: ARB Sequencer is Running<br>Bit 1: ARB Sequencer is Armed and ready for start trigger |
| *pnStatus* | PSHORT | Returned status: 0 on success, negative number on failure. |

### Comments

All channels configured to generate waveform have the size waveform length.

### Example

The following example checks if ARB group B is running

```
DWORD dwStatus

GxAoArbGetGroupStatus(nHandle, GXAO_GROUPB, &dwStatus, &nStatus);
if (dwStatus & 0x1)=1) printf("Group B ARB is running");
```

### See Also

**GxAoArbSetGroupChannels, GxAoArbReadChannelWaveform, GxAoGetErrorString**

## GxAoArbGetGroupStreamingStatus

### Purpose

Returns the group streaming status.

### Applies

**GX1649-1**

### Syntax

**GxAoArbGetGroupStreamingStatus** (*nHandle, nGroup, pwStatus, pwCount, pnStatus*)

### Parameters

| Name | Type | Comments |
|---|---|---|
| *nHandle* | SHORT | Handle to a GX1649 board. |
| *nGroup* | SHORT | Group number:<br>0. GXAO_GROUPA<br>1. GXAO_GROUPB<br>2. GXAO_GROUPC<br>3. GXAO_GROUPD |
| *pwStatus* | PWORD | Returns the a bit field that contains the status of the ARB during streaming operation:<br>Bit 0: The streaming FIFO is half empty<br>Bit 1: The streaming FIFO is empty<br>Bit 2: The streaming FIFO is full |
| *pwCount* | PWORD | Returns the total number of samples currently stored in the streaming FIFO. (0 to 1023) |
| *pnStatus* | PSHORT | Returned status: 0 on success, negative number on failure. |

### Comments

The function returns the status of the streaming FIFO capacity. This can be used to determine when to write to the FIFO when streaming using the polling method (not using interrupts).

### Example

The following example writes to the streaming FIFO when the FIFO sample count is below 500:

```
SHORT     nStatus;
WORD      wStatus, wCount;
DOUBLE    dWaveform[500];
BOOL      bRun=TRUE;

while (bRun)
{
    GxAoArbGetGroupStreamingStatus(nHandle, GXAO_GROUPB, &wStatus, &wCount, &nStatus);
    if (wCount<500)
          GxAoArbWriteStreamingData(nHandle, GXAO_GROUPB, dWaveform, 500,
                      GXAO_1649_WAVEFORM_TYPE_DOUBLE, &nStatus);
}
```

### See Also

**GxAoArbEnableGroupStreaming, GxAoArbWriteStreamingData, GxAoGetErrorString**

## GxAoArbGetGroupTrigger

### Purpose

Returns the specified group trigger mode.

### Applies

**GX1649**

### Syntax

**GxAoArbGetGroupTrigger** (*nHandle, nGroup, pnTriggerMode, pnStatus*)

### Parameters

| Name | Type | Comments |
|---|---|---|
| *nHandle* | SHORT | Handle to a GX1649 board. |
| *nGroup* | SHORT | Group number:<br>0. GXAO_GROUPA<br>1. GXAO_GROUPB<br>2. GXAO_GROUPC<br>3. GXAO_GROUPD |
| *pnTriggerMode* | SHORT | Returned trigger mode:<br>0. GXAO_1649_TRIGGERMODE_SOFTWARE_ONLY<br>1. GXAO_1649_TRIGGERMODE_EXTERNAL<br>2. GXAO_1649_TRIGGERMODE_PXI0<br>3. GXAO_1649_TRIGGERMODE_PXI1<br>4. GXAO_1649_TRIGGERMODE_PXI2<br>5. GXAO_1649_TRIGGERMODE_PXI3<br>6. GXAO_1649_TRIGGERMODE_PXI4<br>7. GXAO_1649_TRIGGERMODE_PXI5<br>8. GXAO_1649_TRIGGERMODE_PXI6<br>9. GXAO_1649_TRIGGERMODE_PXI7<br>10. GXAO_1649_TRIGGERMODE_STAR<br>11. GXAO_1649_TRIGGERMODE_GLOBAL_GROUPS |
| *pnStatus* | PSHORT | Returned status: 0 on success, negative number on failure. |

### Comments

Default trigger mode is software only (GXAO_1649_TRIGGERMODE_SOFTWARE_ONLY). Setting the trigger mode to global (GXAO_1649_TRIGGERMODE_GLOBAL_GROUPS) will allow global software trigger to trigger multiple groups at the same time.

### Example

The following example returns the trigger mode for group A:

```
SHORT nStatus;
SHORT nTriggerMode;

GxAoArbGetGroupTrigger(nHandle, GXAO_GROUPA, &nTriggerMode &nStatus);
```

### See Also

**GxAoArbSetGroupTrigger, GxAoGetErrorString**

## GxAoArbIsGroupStreaming

### Purpose

Returns the group streaming mode.

### Applies

**GX1649-1**

### Syntax

**GxAoArbIsGroupStreaming** (*nHandle, nGroup, pbEnable, pnStatus*)

### Parameters

| Name | Type | Comments |
|---|---|---|
| *nHandle* | SHORT | Handle to a GX1649-1 board. |
| *nGroup* | SHORT | Group number:<br>0. GXAO_GROUPA<br>1. GXAO_GROUPB<br>2. GXAO_GROUPC<br>3. GXAO_GROUPD |
| *pbEnable* | PBOOL | Returns TRUE if ARB Group is in streaming mode and FALSE if ARB group is in non-streaming mode. |
| *pnStatus* | PSHORT | Returned status: 0 on success, negative number on failure. |

### Comments

The GX1649-1 ARB can operate in streaming or non-streaming mode. By default, each group (Group A-D) is set to non-streaming mode. When the board is in non-streaming mode, the ARB will use the onboard memory as a source for the waveform it will generate. When the board is in streaming mode, the ARB will continuously generate a waveform. The PC will write samples to a FIFO located on the ARB as needed to maintain the streaming operation.

### Example

The following example gets group B mode streaming state:

```
SHORT nStatus;
BOOL  bEnable;
GxAoArbIsGroupStreaming (nHandle, GXAO_GROUPB, &bEnable, &nStatus);
if (bEnable)
    printf("ARB Group B is in streaming mode");
```

### See Also

**GxAoArbEnableGroupStreaming, GxAoArbGetGroupStreamingStatus, GxAoArbSetupStreamingInterrupt, GxAoGetErrorString**

## GxAoArbReadChannelWaveform

### Purpose

Returns the specified channel waveform to an array.

### Applies

**GX1649**

### Syntax

**GxAoArbReadChannelWaveform** (*nHandle, nGroup, nChannel, pvWaveform, pdwWaveformSize, nWaveformType, pnStatus*)

### Parameters

| Name | Type | Comments |
|---|---|---|
| *nHandle* | SHORT | Handle to a GX1649 board. |
| *nGroup* | SHORT | Group number:<br>0. GXAO_GROUPA<br>1. GXAO_GROUPB<br>2. GXAO_GROUPC<br>3. GXAO_GROUPD |
| *nChannel* | PSHORT | Channel number (0-15) |
| *pvWaveform* | PVOID | Buffer to hold the returned waveform array of Words or Doubles (see *nWaveformType*). |
| *pdwWaveformSize* | PDWORD | When calling this function it should hold the number of elements in *pvWaveform*. On return, the function will return the actual number of elements copied to the array. |
| *nWaveformType* | SHORT | The type of waveform data to write:<br>0. GXAO_1649_WAVEFORM_TYPE_WORD<br>1. GXAO_1649_WAVEFORM_TYPE_DOUBLE |
| *pnStatus* | PSHORT | Returned status: 0 on success, negative number on failure. |

### Example

The following example returns the clock source and frequency for group A:

```
SHORT  nStatus;
DOUBLE adWave[]={1.0, 2.0, 3.0, 4.0};
DWORD  dwSize=4;

GxAoArbWriteChannelWaveform(nHandle, GXAO_GROUPA, 0, adWave, dwSize,
                      GXAO_1649_WAVEFORM_TYPE_DOUBLE, &nStatus);
GxAoArbReadChannelWaveform(nHandle, GXAO_GROUPA, 0, adWave, &dwSize,
                      GXAO_1649_WAVEFORM_TYPE_DOUBLE, &nStatus);
```

### See Also

**GxAoArbSetGroupChannels** , **GxAoArbWriteChannelWaveform, GxAoGetErrorString**

## GxAoArbResumeStreamingInterrupt

### Purpose

Resume processing interrupts after interrupt handler is called

### Applies

**GX1649-1**

### Syntax

**GxAoArbResumeStreamingInterrupt** (*nHandle, pnStatus*)

### Parameters

| Name | Type | Comments |
|---|---|---|
| *nHandle* | SHORT | Handle to a GX1649 board. |
| *pnStatus* | PSHORT | Returned status: 0 on success, negative number on failure. |

### Comments

When the board generates an interrupt, the kernel will automatically block new interrupts from calling the interrupt handler until after this function is called. This function should be called at the end of the interrupt handler, so that the next interrupt can be processed when the FIFO is half empty.

### Example

The following example shows interrupt handler for streaming, after data is written to the FIFO, **GxAoArbSetupStreamingInterrupt** is called at the end after handling is done.

```
LONG WINAPI InterruptCallback(SHORT nHandle, SHORT enEventType, PVOID pvUserData)
{
    static int s_iInterruptCount=0;
    short nStatus;
    WORD wStatus, wCount;
    INT iGroup;
    BOOL bEnable;

    // search all group for source of interrupt
    for (iGroup=GXAO_GROUPA ; iGroup<=GXAO_GROUPD ; iGroup++)
    {   // ignore group if streaming is not enabled
        GxAoArbIsGroupStreaming(nHandle, iGroup, &bEnable, &nStatus);
        CheckStatus(nStatus);
        if (!bEnable) continue;
        // retrieve streaming FIFO status
        GxAoArbGetGroupStreamingStatus(nHandle, iGroup, &wStatus, &wCount, &nStatus);
        CheckStatus(nStatus);
        // if FIFO is at least half empty
        if ((wStatus & GXAO_1649_ARB_STREAMING_STATUS_FULL)==0)
        {       printf("Group%c Interrupt received %d\r\n", 'A'+iGroup, ++s_iInterruptCount);
                // alternate output between sine and square wave
                if (s_iInterruptCount & 1)
                    GxAoArbWriteStreamingData(nHandle, iGroup, &g_adWaveformData[0],
                        POINTS_PER_WAVEFORM, GXAO_1649_WAVEFORM_TYPE_DOUBLE, &nStatus);
                else
```

```
                GxAoArbWriteStreamingData(nHandle, iGroup,
                       &g_adWaveformData[POINTS_PER_WAVEFORM],
                       POINTS_PER_WAVEFORM, GXAO_1649_WAVEFORM_TYPE_DOUBLE, &nStatus);
               CheckStatus(nStatus);
        }
    }

    // resume streaming interrupt
    GxAoArbResumeStreamingInterrupt(nHandle, &nStatus);
    return 0;
}
```

## See Also

**GxAoArbEnableGroupStreaming, GxAoArbSetupStreamingInterrupt, GxAoArbEnableGroupStreaming,, xAoArbWriteStreamingData, GxAoGetErrorString**

## GxAoArbSetGroupChannels

### Purpose

Sets the group channels used for arbitrary waveform generation and the waveform size.

### Applies

**GX1649**

### Syntax

**GxAoArbSetGroupChannels** (*nHandle, nGroup, pwChannels, dwWaveformSize, pnStatus*)

### Parameters

| Name | Type | Comments |
|---|---|---|
| *nHandle* | SHORT | Handle to a GX1649 board. |
| *nGroup* | SHORT | Group number:<br>0. GXAO_GROUPA<br>1. GXAO_GROUPB<br>2. GXAO_GROUPC<br>3. GXAO_GROUPD |
| *wChannels* | WORD | A bit mask where each bit corresponds to a channel (bit 0 is channel 0, bit 15 channel 15). Bits that are set to 1 indicate that the corresponding channels are set to generate waveform. Bits that are set to 0 indicate a static output channel.. |
| *dwWaveformSize* | DWORD | Waveform size when in non-streaming mode. This parameter is ignored when in streaming mode |
| *pnStatus* | PSHORT | Returned status: 0 on success, negative number on failure. |

### Comments

All channels configured to generate waveform have the size waveform length. The waveform maximum size depends on the number of channels that are configured to generate waveform. Total group memory is 256K elements. If all channels are configured to generate waveforms (*wChannels*=0xFFFF) than the maximum waveform length is 16K (256K/16).

### Example

The following example sets channel 1, 3 and 15 (0x800A) in group B to generate waveform with the length to 16K:

```
SHORT nStatus;
GxAoArbSetGroupChannels(nHandle, GXAO_GROUPB, 0x800A, 16384, &nStatus);
```

### See Also

**GxAoArbGetGroupChannels, GxAoArbWriteChannelWaveform, GxAoGetErrorString**

## GxAoArbSetGroupClock

### Purpose

Sets the specified group clock source and frequency.

### Applies

**GX1649**

### Syntax

**GxAoArbSetGroupClock** (*nHandle, nGroup, nClockSource, dFrequency, pnStatus*)

### Parameters

| Name | Type | Comments |
|---|---|---|
| *nHandle* | SHORT | Handle to a GX1649 board. |
| *nGroup* | SHORT | Group number:<br>0. GXAO_GROUPA<br>1. GXAO_GROUPB<br>2. GXAO_GROUPC<br>3. GXAO_GROUPD |
| *nClockSource* | SHORT | Clock source:<br>0. GXAO_1649_CLOCKSOURCE_INTERNAL<br>1. GXAO_1649_CLOCKSOURCE_EXTERNAL<br>2. GXAO_1649_CLOCKSOURCE_PXI0<br>3. GXAO_1649_CLOCKSOURCE_PXI1<br>4. GXAO_1649_CLOCKSOURCE_PXI2<br>5. GXAO_1649_CLOCKSOURCE_PXI3<br>6. GXAO_1649_CLOCKSOURCE_PXI4<br>7. GXAO_1649_CLOCKSOURCE_PXI5<br>8. GXAO_1649_CLOCKSOURCE_PXI6<br>9. GXAO_1649_CLOCKSOURCE_PXI7<br>10. GXAO_1649_CLOCKSOURCE_STAR<br>11. GXAO_1649_CLOCKSOURCE_GROUPA |
| *dFrequncy* | DOUBLE | Group frequency, this is only relevant when pnClockSource is control by the card internally (GXAO_1649_CLOCKSOURCE_INTERNAL). Frequency range is 1-10MHz) |
| *pnStatus* | PSHORT | Returned status: 0 on success, negative number on failure. |

### Comments

Default clock source is GXAO_1649_CLOCKSOURCE_INTERNAL with frequency of 10MHz
GXAO_1649_CLOCKSOURCE_GROUPA can only be used when *nGroup* is set to B, C, and D, using group A clock source will synchronizes all group channels to use the same clock source as group A

### Example

The following example sets group A with clock source of internal with frequency of 5MHz:

```
SHORT nStatus;

GxAoArbSetGroupClock (nHandle, GXAO_GROUPA, GXAO_1649_CLOCKSOURCE_INTERNAL, 5000000, &nStatus);
```

### See Also

GxAoArbGetGroupClock, GxAoGetErrorString

## GxAoArbSetGroupTrigger

### Purpose

Sets the specified group trigger mode.

### Applies

**GX1649**

### Syntax

**GxAoArbSetGroupTrigger** (*nHandle, nGroup, nTriggerMode, pnStatus*)

### Parameters

| Name | Type | Comments |
|---|---|---|
| *nHandle* | SHORT | Handle to a GX1649 board. |
| *nGroup* | SHORT | Group number:<br>0. GXAO_GROUPA<br>1. GXAO_GROUPB<br>2. GXAO_GROUPC<br>3. GXAO_GROUPD |
| *nTriggerMode* | SHORT | Trigger mode:<br>0. GXAO_1649_TRIGGERMODE_SOFTWARE_ONLY<br>1. GXAO_1649_TRIGGERMODE_EXTERNAL<br>2. GXAO_1649_TRIGGERMODE_PXI0<br>3. GXAO_1649_TRIGGERMODE_PXI1<br>4. GXAO_1649_TRIGGERMODE_PXI2<br>5. GXAO_1649_TRIGGERMODE_PXI3<br>6. GXAO_1649_TRIGGERMODE_PXI4<br>7. GXAO_1649_TRIGGERMODE_PXI5<br>8. GXAO_1649_TRIGGERMODE_PXI6<br>9. GXAO_1649_TRIGGERMODE_PXI7<br>10. GXAO_1649_TRIGGERMODE_STAR<br>11. GXAO_1649_TRIGGERMODE_GLOBAL |
| *pnStatus* | PSHORT | Returned status: 0 on success, negative number on failure. |

### Comments

Default trigger mode is software only (GXAO_1649_TRIGGERMODE_SOFTWARE_ONLY). Setting the trigger mode to global (GXAO_1649_TRIGGERMODE_GLOBAL) will allow global software trigger to trigger multiple groups at the same time.

### Example

The following example sets the trigger mode for group A to external trigger:

```
SHORT nStatus;

GxAoArbSetGroupTrigger(nHandle, GXAO_GROUPA, GXAO_1649_TRIGGERMODE_EXTERNAL, &nStatus);
```

### See Also

**GxAoArbGetGroupTrigger, GxAoArbTrigGroup, GxAoGetErrorString**

## GxAoArbSetupStreamingInterrupt

### Purpose

Configures the ARB streaming interrupt by setting which ARB groups are used as a source for an interrupt, and setting the interrupt call back function and its optional parameter.

### Applies

**GX1649-1**

### Syntax

**GxAoArbSetupStreamingInterrupt** (*nHandle, wGroups, pInterruptCallback, pvInterruptCallbackParam, pnStatus*)

### Parameters

| Name | Type | Comments |
|---|---|---|
| *nHandle* | SHORT | Handle to a GX1649-1 board. |
| *wGroups* | WORD | Set which groups can cause an interrupt, encoded in 4 bits of wGroups, bit 0 corresponds to Group A and bit 3 corresponds to Group D |
| *pInterruptCallback* | Gt_EventCallback | The user defined callback function for the interrupt handler |
| *pvInterruptCallbackParam* | LPVOID | Optional parameter to be passed to the user defined call back function defined by the *pInterruptCallback* parameter |
| *pnStatus* | PSHORT | Returned status: 0 on success, negative number on failure. |

### Comments

This function sets the board up for streaming operation. This function is used to select which ARB groups will cause an interrupt to be generated. When an ARB group is in streaming mode, it will cause the board to generate a PCI interrupt when the streaming FIFO is half full. This function assigns a user defined call back function (**pInterruptCallback**) to the PCI interrupt. The user defined callback function must use the following function prototype:

**LONG CallBackFunction (SHORT nHandle, SHORT enEventType, LPVOID pvUserData)**

### Example

The following example configures the streaming interrupt to use group C ARB and set the interrupt handler to the user defined function CallBackFunction with no optional parameter:

```
SHORT nStatus;

GxAoArbSetupStreamingInterrupt (nHandle, 0x4, CallBackFunction, NULL, &nStatus);
```

### See Also

**GxAoArbEnableGroupStreaming, GxAoArbResumeStreamingInterrupt, GxAoArbWriteStreamingData, GxAoGetErrorString**

## GxAoArbTrigGroup

### Purpose

Trigger the group and starts waveform generation.

### Applies

**GX1649**

### Syntax

**GxAoArbTrigGroup** (*nHandle, nGroup, pnStatus*)

### Parameters

| Name | Type | Comments |
|---|---|---|
| *nHandle* | SHORT | Handle to a GX1649 board. |
| *nGroup* | SHORT | Group number:<br>0. GXAO_GROUPA<br>1. GXAO_GROUPB<br>2. GXAO_GROUPC<br>3. GXAO_GROUPD |
| *pnStatus* | PSHORT | Returned status: 0 on success, negative number on failure. |

### Comments

This function will start generating the waveform in all trigger modes. The group must be in armed state.

### Example

The following start the waveform generation:

```
SHORT nStatus;

GxAoArbArmGroup (nHandle, GXAO_GROUPA, TRUE, TRUE, &nStatus);
GxAoArbTrigGroup (nHandle, GXAO_GROUPA, &nStatus);
```

### See Also

**GxAoArbArmGroup, GxAoArbSetGroupTrigger, GxAoGetErrorString**

## GxAoArbWriteChannelWaveform

### Purpose

Sets the specified channel waveform.

### Applies

**GX1649**

### Syntax

**GxAoArbWriteChannelWaveform** (*nHandle, nGroup, nChannel,* p*vWaveform ,dwWaveformSize, nWaveformType, pnStatus*)

### Parameters

| Name | Type | Comments |
|---|---|---|
| *nHandle* | SHORT | Handle to a GX1649 board. |
| *nGroup* | SHORT | Group number:<br>0. GXAO_GROUPA<br>1. GXAO_GROUPB<br>2. GXAO_GROUPC<br>3. GXAO_GROUPD |
| *nChannel* | PSHORT | Channel number (0-15) |
| *pvWaveform* | PVOID | Buffer to hold the returned waveform array of Words or Doubles (see *nWaveformType*). |
| *dwWaveformSize* | DWORD | Number of elements to in the buffer to write (1 to x, where x is 256K divided by number of channels that are set to generate waveform). |
| *nWaveformType* | SHORT | The type of waveform data to write:<br>0. GXAO_1649_WAVEFORM_TYPE_WORD<br>1. GXAO_1649_WAVEFORM_TYPE_DOUBLE |
| *pnStatus* | PSHORT | Returned status: 0 on success, negative number on failure. |

### Comments

The waveform maximum size depends on the number of channels that are configured to generate waveform (see **GxAoArbSetGroupChannels)**. Total group memory is 256K elements. If all channels are configured to generate waveforms than the maximum waveform length is 16K (256K/16).

### Example

The following example writes the waveform for group A channel 7 in Double data type format:

```
SHORT  nStatus;
DOUBLE adWave[]={1.0, 2.0, 3.0, 4.0};

GxAoArbWriteChannelWaveform(nHandle, GXAO_GROUPA, 7, adWave, 4, GXAO_1649_WAVEFORM_TYPE_DOUBLE,
                        &nStatus);
```

### See Also

**GxAoArbReadChannelWaveform, GxAoArbSetGroupChannels, GxAoGetErrorString**

## GxAoArbWriteStreamingData

### Purpose

Write to the streaming FIFO when in streaming mode

### Applies

**GX1649-1**

### Syntax

**GxAoArbWriteStreamingData** (*nHandle, nGroup, pvWaveform, dwSamples, nWaveformType, pnStatus*)

### Parameters

| Name | Type | Comments |
|---|---|---|
| *nHandle* | SHORT | Handle to a GX1649-1 board. |
| *nGroup* | SHORT | Group number:<br>0. GXAO_GROUPA<br>1. GXAO_GROUPB<br>2. GXAO_GROUPC<br>3. GXAO_GROUPD |
| *pvWaveform* | PVOID | Buffer to hold the returned waveform array of Words or Doubles (see *nWaveformType*). |
| *dwSamples* | DWORD | Number of elements to write to the streaming FIFO (1 to 1024) |
| *nWaveformType* | SHORT | The type of waveform data to write:<br>0. GXAO_1649_WAVEFORM_TYPE_WORD<br>1. GXAO_1649_WAVEFORM_TYPE_DOUBLE |
| *pnStatus* | PSHORT | Returned status: 0 on success, negative number on failure. |

### Comments

The waveform maximum size depends on the number of channels that are configured to generate waveform (see **GxAoArbSetGroupChannels)**. Total streaming FIFO memory is 1K elements. The FIFO can be written to while the ARB sequencer is running. This allows a continuous ARB operation and periodic calls to this function to add more samples to the FIFO as the sequencer empties its contents.

If using multiple channels, the waveform array must contain samples for each channel. For example, if channels 0, 5, and 7 are configured as ARB (see **GxAoArbSetGroupChannels**) then the waveform array should contain samples as follows: {Ch 0 Sample 0, Ch 5 Sample 0, Ch7 Sample 0, Ch 0 Sample 1, Ch 5 Sample 1,Ch 7 Sample 1, Ch 0 Sample 2,...}

### Example

The following example writes 4 samples to the Group A ARB streaming FIFO in Double data type format. Channels 0 and channel 1 are participating in the streaming ARB:

```
SHORT  nStatus;
//Channel 0 samples are 1.0V and 3.0V
//Channel 1 samples are 2.0V and 4.0V
DOUBLE adWave[]={1.0, 2.0, 3.0, 4.0};

GxAoArbSetGroupChannels(nHandle, GXAO_GROUPA, 0x3, 1, &nStatus);
GxAoArbWriteChannelWaveform(nHandle, GXAO_GROUPA, adWave, 4, GXAO_1649_WAVEFORM_TYPE_DOUBLE,
                        &nStatus);
```

### See Also

**GxAoArbWriteChannelWaveform, GxAoArbSetGroupChannels, GxAoGetErrorString**

## GxAoCalADC

### Purpose

Calibrates the onboard ADC

### Applies

**GX1649**

### Syntax

**GxAoCalADC** (*nHandle, dRefPositiveMeasurement, dRefNegativeMeasurement, dADCPositiveMeasurement, dADCNegativeMeasurement, pnStatus*)

### Parameters

| Name | Type | Comments |
|---|---|---|
| *nHandle* | SHORT | Handle to a GX1649 board. |
| *dRefPositiveMeasurement* | DOUBLE | Positive Calibration measurement from external reference DMM |
| *dRefNegativeMeasurement* | DOUBLE | Negative Calibration measurement from external reference DMM |
| *dADCPositiveMeasurement* | DOUBLE | Positive Calibration measurement from onboard ADC |
| *dADCNegativeMeasurement* | DOUBLE | Negative Calibration measurement from onboard ADC |
| *pnStatus* | PSHORT | Returned status: 0 on success, negative number on failure. |

### Comments

The first step in calibrating the GX1649 is to calibrate the onboard ADC. In order to do this, an external DMM is required. The ADC is calibrated using two voltage points, +15V and -15V generated on Group A, Channel 0. For each voltage point, Group A Channel 0 is measured using the external DMM and the onboard ADC (using the **GxAoCalADCMeasure** function).

After all four measurements are taken (two measurements from the onboard ADC and two measurements from the external reference DMM), this function should be called and the measurements passed in as parameters.

### Example

The following example calibrates the onboard ADC using an external DMM:

```
SHORT nStatus;
DOUBLE dRefPositiveMeasurement, dRefNegativeMeasurement, dADCPositiveMeasurement,
                      dADCNegativeMeasurement;

GxAoCalSetVoltagePoint (nHandle, GXAO_GROUPA, 0, GX1649_CAL_VOLTAGE_POINT_NEGATIVE, &nStatus);
Sleep(1000);
//External DMM Function, substitute with actual External DMM call
DmmMeasure(&dRefNegativeMeasurement);
GxAoCalADCMeasure (nHandle, GXAO_GROUPA, 0, &dADCNegativeMeasurement &nStatus);

GxAoCalSetVoltagePoint (nHandle, GXAO_GROUPA, 0, GX1649_CAL_VOLTAGE_POINT_POSITIVE, &nStatus);
Sleep(1000);
//External DMM Function, substitute with actual External DMM call
DmmMeasure(&dRefPositiveMeasurement);
GxAoCalADCMeasure (nHandle, GXAO_GROUPA, 0, &dADCPositiveMeasurement &nStatus);

// set the ADC calibration measurements
GxAoCalADC(nHadle, dRefPositiveMeasurement, dRefNegativeMeasurement, dADCPositiveMeasurement ,
                      dADCNegativeMeasurement, &nStatus);
```

```
//Finalize calibration by writing calibration record to EEPROM
GxAoCalWriteEEPROM(nHandle, &nStatus);
```

### See Also

**GxAoCalADCMeasure, GxAoCalSetVoltagePoint, GxAoCalWriteEEPROM** , **GxAoGetErrorString**

## GxAoCalADCMeasure

### Purpose

Returns a measurement from the onboard ADC (used for calibration of the ADC)

### Applies

**GX1649**

### Syntax

**GxAoCalADCMeasure** (*nHandle, nGroup, nChannel, pdMeasurement, pnStatus*)

### Parameters

| Name | Type | Comments |
|---|---|---|
| *nHandle* | SHORT | Handle to a GX1649 board. |
| *nGroup* | SHORT | Group number:<br>0. GXAO_GROUPA<br>1. GXAO_GROUPB<br>2. GXAO_GROUPC<br>3. GXAO_GROUPD |
| *nChannel* | SHORT | Channel number (0-15) |
| *pdMeasurement* | PDOUBLE | Returns measurement from ADC |
| *pnStatus* | PSHORT | Returned status: 0 on success, negative number on failure. |

### Comments

The first step in calibrating the GX1649 is to calibrate the onboard ADC. In order to do this, an external DMM is required. This function is used to take baseline measurements from the onboard ADC for the two different voltage points during calibration. These measurements, along with measurements taken from an external DMM are then used to calculate a gain and offset for the onboard ADC.

### Example

The following example calibrates the onboard ADC using an external DMM:

```
SHORT nStatus;
DOUBLE dRefPositiveMeasurement, dRefNegativeMeasurement, dADCPositiveMeasurement,
                      dADCNegativeMeasurement;

GxAoCalSetVoltagePoint (nHandle, GXAO_GROUPA, 0, GX1649_CAL_VOLTAGE_POINT_NEGATIVE, &nStatus);
Sleep(1000);
//External DMM Function, substitute with actual External DMM call
DmmMeasure(&dRefNegativeMeasurement);
GxAoCalADCMeasure (nHandle, GXAO_GROUPA, 0, &dADCNegativeMeasurement &nStatus);

GxAoCalSetVoltagePoint (nHandle, GXAO_GROUPA, 0, GX1649_CAL_VOLTAGE_POINT_POSITIVE, &nStatus);
Sleep(1000);
//External DMM Function, substitute with actual External DMM call
DmmMeasure(&dRefPositiveMeasurement);
GxAoCalADCMeasure (nHandle, GXAO_GROUPA, 0, &dADCPositiveMeasurement &nStatus);

// set the ADC calibration measurements
GxAoCalADC(nHadle, dRefPositiveMeasurement, dRefNegativeMeasurement, dADCPositiveMeasurement ,
                      dADCNegativeMeasurement, &nStatus);

//Finalize calibration by writing calibration record to EEPROM
GxAoCalWriteEEPROM(nHandle, &nStatus);
```

### See Also

**GxAoCalADC, GxAoCalSetVoltagePoint, GxAoCalWriteEEPROM , GxAoGetErrorString**

## GxAoCalGroupChannel

### Purpose

Calibrates a channel within a group using the onboard ADC

### Applies

**GX1649**

### Syntax

**GxAoCalGroupChannel** (*nHandle, nGroup, nChannel, dRefPositiveMeasurement, dRefNegativeMeasurement, pnStatus*)

### Parameters

| Name | Type | Comments |
|---|---|---|
| *nHandle* | SHORT | Handle to a GX1649 board. |
| *nGroup* | SHORT | Group number to calibrate:<br>0. GXAO_GROUPA<br>1. GXAO_GROUPB<br>2. GXAO_GROUPC<br>3. GXAO_GROUPD |
| *nChannel* | SHORT | Channel within a group to calibrate (0-15) |
| *dRefPositiveMeasurement* | DOUBLE | Positive measurement taken by reference DMM |
| *dRefNegativeMeasurement* | DOUBLE | Negative measurement taken by reference DMM |
| *pnStatus* | PSHORT | Returned status: 0 on success, negative number on failure. |

### Comments

This function calibrates a group's channel to within 1mV of accuracy.

### Example

The following example calibrates all groups (A to D) and channels (0-15):

```
SHORT nStatus, nGroup, nChannel;
DOUBLE dRefPositiveMeasurement, dRefNegativeMeasurement;

for(nGroup=GXAO_GROUPA; nGroup<=GXAO_GROUPD; nGroup++)
{
       for(nChannel=0; nChannel<64; nChannel++)
{
        GxAoCalSetVoltagePoint (nHandle, nGrouop, nChannel, GX1649_CAL_VOLTAGE_POINT_POSITIVE,
                        &nStatus);
        DMMMeasure(&dRefPositiveMeasurement);

        GxAoCalSetVoltagePoint (nHandle, nGroup, nChannel, GX1649_CAL_VOLTAGE_POINT_NEGATIVE,
                        &nStatus);
        DMMMeasure(&dRefNegativeMeasurement);
GxAoCalGroupChannel(nHandle, nGroup, nChannel, dRefPositiveMeasurement, dNegativeMeasurement,
                        &nStatus);
        if(nStatus!=0)
                        printf("Error In Calibration!");
}
}
//Finalize calibration by writing calibration record to EEPROM
GxAoCalWriteEEPROM(nHandle, &nStatus);
```

### See Also

**GxAoCalADC, GxAoCalWriteEEPROM , GxAoGetErrorString**

## GxAoCalSetMode

### Purpose

Sets the driver in calibration mode

### Applies

**GX1649**

### Syntax

**GxAoCalSetMode** (*nHandle*, *nMode*, *pnStatus*)

### Parameters

| Name | Type | Comments |
|---|---|---|
| *nHandle* | SHORT | Handle to a GX1649 board. |
| *nMode* | SHORT | Group number to calibrate:<br>GX1649_CAL_MODE_DISABLED<br>0. GX1649_CAL_MODE_ENABLED |
| *pnStatus* | PSHORT | Returned status: 0 on success, negative number on failure. |

### Comments

This function must be called before calling any one of the calibration functions. You must have a valid calibration license installed otherwise this function will fail.

### Example

The following example calibrates the onboard ADC using an external DMM:

```
SHORT nStatus;
DOUBLE dRefPositiveMeasurement, dRefNegativeMeasurement, dADCPositiveMeasurement,
                      dADCNegativeMeasurement;

GxAoCalSetMode (nHandle, GXAO_CAL_MODE_ENABLED, &nStatus);
if (nStatus<0)
{                     printf("Unable to enter calibration mode. Aborting….");
                      return;
}

GxAoCalSetVoltagePoint (nHandle, GXAO_GROUPA, 0, GX1649_CAL_VOLTAGE_POINT_NEGATIVE, &nStatus);
Sleep(1000);
//External DMM Function, substitute with actual External DMM call
DmmMeasure(&dRefNegativeMeasurement);
GxAoCalADCMeasure (nHandle, GXAO_GROUPA, 0, &dADCNegativeMeasurement &nStatus);
GxAoCalSetVoltagePoint (nHandle, GXAO_GROUPA, 0, GX1649_CAL_VOLTAGE_POINT_POSITIVE, &nStatus);
Sleep(1000);
//External DMM Function, substitute with actual External DMM call
DmmMeasure(&dRefPositiveMeasurement);
GxAoCalADCMeasure (nHandle, GXAO_GROUPA, 0, &dADCPositiveMeasurement &nStatus);

//Finalize calibration by writing calibration record to EEPROM
GxAoCalADC(nHandle, dRefPositiveMeasurement, dRefNegativeMeasurement, dADCPositiveMeasurement,
                      dADCNegativeMeasurement, &nStatus);


GxAoCalWriteEEPROM(nHandle, &nStatus);
```

### See Also

**GxAoCalADCMeasure, GxAoCalADC, GxAoCalGroupChannel , GxAoCalWriteEEPROM, GxAoGetErrorString**

## GxAoCalSetVoltagePoint

### Purpose

Sets the output voltage for Channel and ADC calibration

### Applies

**GX1649**

### Syntax

**GxAoCalSetVoltagePoint** (*nHandle, nGroup, nChannel, nVoltagePoint, pnStatus*)

### Parameters

| Name | Type | Comments |
|---|---|---|
| *nHandle* | SHORT | Handle to a GX1649 board. |
| *nGroup* | SHORT | Group number to calibrate:<br>1. GXAO_GROUPA<br>2. GXAO_GROUPB<br>3. GXAO_GROUPC<br>4. GXAO_GROUPD |
| *nChannel* | SHORT | Channel within a group to calibrate (0-15) |
| *nVoltagePoint* | SHORT | Sets the current calibration voltage point:<br>0. GX1649_CAL_VOLTAGE_POINT_NEGATIVE (-15 Volts)<br>1. GX1649_CAL_VOLTAGE_POINT_POSITIVE (+15 Volts) |
| *pnStatus* | PSHORT | Returned status: 0 on success, negative number on failure. |

### Comments

This function sets the output voltage on a channel to a calibration voltage point. Each calibration voltage point must be measured using an external reference DMM or the onboard ADC.

### Example

The following example calibrates the onboard ADC using an external DMM:

```
SHORT nStatus;
DOUBLE dRefPositiveMeasurement, dRefNegativeMeasurement, dADCPositiveMeasurement,
                      dADCNegativeMeasurement;

GxAoCalSetVoltagePoint (nHandle, GXAO_GROUPA, 0, GX1649_CAL_VOLTAGE_POINT_NEGATIVE, &nStatus);
Sleep(1000);
//External DMM Function, substitute with actual External DMM call
DmmMeasure(&dRefNegativeMeasurement);
GxAoCalADCMeasure (nHandle, GXAO_GROUPA, 0, &dADCNegativeMeasurement &nStatus);
GxAoCalSetVoltagePoint (nHandle, GXAO_GROUPA, 0, GX1649_CAL_VOLTAGE_POINT_POSITIVE, &nStatus);
Sleep(1000);
//External DMM Function, substitute with actual External DMM call
DmmMeasure(&dRefPositiveMeasurement);
GxAoCalADCMeasure (nHandle, GXAO_GROUPA, 0, &dADCPositiveMeasurement &nStatus);

//Finalize calibration by writing calibration record to EEPROM
GxAoCalADC(nHandle, dRefPositiveMeasurement, dRefNegativeMeasurement, dADCPositiveMeasurement,
                      dADCNegativeMeasurement, &nStatus);


GxAoCalWriteEEPROM(nHandle, &nStatus);
```

### See Also

**GxAoCalADCMeasure, GxAoCalADC, GxAoCalGroupChannel , GxAoCalWriteEEPROM, GxAoGetErrorString**

## GxAoCalWriteEEPROM

### Purpose

Finalize calibration of the onboard ADC and each channel by writing to the EEPROM.

### Applies

**GX1649**

### Syntax

**GxAoCalWriteEEPROM** (*nHandle*, *pnStatus*)

### Parameters

| Name | Type | Comments |
|---|---|---|
| *nHandle* | SHORT | Handle to a GX1649 board. |
| *pnStatus* | PSHORT | Returned status: 0 on success, negative number on failure. |

### Comments

This function finalizes the calibration by writing the calibration constants to the onboard EEPROM.

### Example

The following example calibrates the onboard ADC using an external DMM:

```
SHORT nStatus;
DOUBLE dRefPositiveMeasurement, dRefNegativeMeasurement, dADCPositiveMeasurement,
                      dADCNegativeMeasurement;

GxAoCalSetVoltagePoint (nHandle, GXAO_GROUP0, 0, GX1649_CAL_VOLTAGE_POINT_NEGATIVE, &nStatus);
Sleep(1000);
//External DMM Function, substitute with actual External DMM call
DmmMeasure(&dRefNegativeMeasurement);
GxAoCalADCMeasure (nHandle, GXAO_GROUP0, 0, &dADCNegativeMeasurement &nStatus);

GxAoCalSetVoltagePoint (nHandle, GXAO_GROUP0, 0, GX1649_CAL_VOLTAGE_POINT_POSITIVE, &nStatus);
Sleep(1000);
//External DMM Function, substitute with actual External DMM call
DmmMeasure(&dRefPositiveMeasurement);
GxAoCalADCMeasure (nHandle, GXAO_GROUP0, 0, &dADCPositiveMeasurement &nStatus);
//Finalize calibration by writing calibration record to EEPROM
GxAoCalWriteEEPROM(nHandle, &nStatus);
```

### See Also

**GxAoCalADCMeasure, GxAoGetErrorString**

## GxAoDioArm

### Purpose

Arms or disables arming the specified group.

### Applies

**GX1649**

### Syntax

**GxAoDioArm** (*nHandle*, *bArm*, *bContinuous*, *pnStatus*)

### Parameters

| Name | Type | Comments |
|---|---|---|
| *nHandle* | SHORT | Handle to a GX1649 board. |
| *bArm* | BOOL | TRUE (1) to arm the group and FALSE (0) to disable arming and stop generating waveforms. |
| *bContinuous* | BOOL | TRUE (1) will cause the waveform to be repeated (until *bArm* is FALSE). FALSE (0) will cause only one waveform to be generated. |
| *pnStatus* | PSHORT | Returned status: 0 on success, negative number on failure. |

### Comments

Disabling arming the group (*bArm*=FALSE) will cause the card to stop generating waveforms.

### Example

The following example generates a repeated waveform for 1000 mSec and then halts:

```
SHORT nStatus;
SHORT nTriggerMode;

GxAoDioArm (nHandle, GXAO_GROUPA, TRUE, TRUE, &nStatus);
GxAoDioTrig(nHandle, &nStatus);
Sleep(1000)
GxAoDioArm (nHandle, GXAO_GROUPA, FALSE, TRUE, &nStatus);
```

### See Also

**GxAoDioTrig, GxAoGetErrorString**

## GxAoDioGetClock

### Purpose

Returns the DIO clock source and frequency.

### Applies

**GX1649**

### Syntax

**GxAoDioGetClock** (*nHandle, pnClockSource, pdFrequency, pnStatus*)

### Parameters

| Name | Type | Comments |
|---|---|---|
| *nHandle* | SHORT | Handle to a GX1649 board. |
| *pnClockSource* | PSHORT | Returned clock source:<br>0. GXAO_1649_CLOCKSOURCE_INTERNAL<br>1. GXAO_1649_CLOCKSOURCE_EXTERNAL<br>2. GXAO_1649_CLOCKSOURCE_PXI0<br>3. GXAO_1649_CLOCKSOURCE_PXI1<br>4. GXAO_1649_CLOCKSOURCE_PXI2<br>5. GXAO_1649_CLOCKSOURCE_PXI3<br>6. GXAO_1649_CLOCKSOURCE_PXI4<br>7. GXAO_1649_CLOCKSOURCE_PXI5<br>8. GXAO_1649_CLOCKSOURCE_PXI6<br>9. GXAO_1649_CLOCKSOURCE_PXI7<br>10. GXAO_1649_CLOCKSOURCE_STAR<br>11. GXAO_1649_CLOCKSOURCE_GROUPA |
| *pdFrequncy* | PDOUBLE | Returned DIO frequency, this is only relevant when pnClockSource is control by the card internally (GXAO_1649_CLOCKSOURCE_INTERNAL) |
| *pnStatus* | PSHORT | Returned status: 0 on success, negative number on failure. |

### Comments

Default clock source is GXAO_1649_CLOCKSOURCE_INTERNAL with frequency of 10MHz GXAO_1649_CLOCKSOURCE_GROUPA can only be used when *nGroup* is set to B, C, and D, using group A clock source will synchronizes all group channels to use the same clock source as group A.

### Example

The following example returns the clock source and frequency for the DIO

```
SHORT nStatus;
SHORT nClockSource;
DOUBLE dFrequncy;

GxAoDioGetClock(nHandle, &nClockSource, &dFrequency, &nStatus);
```

### See Also

**GxAoDioSetClock, GxAoGetErrorString**

## GxAoDioGetOutputEnable

### Purpose

Returns the Digital IO output enable states

### Applies

**GX1649**

### Syntax

**GxAoDioGetOutputEnable**(*nHandle, pucOutputEnable, pnStatus*)

### Parameters

| Name | Type | Comments |
|---|---|---|
| *nHandle* | SHORT | Handle to a GX1649 board. |
| *pucOutputEnable* | PBYTE | Returns the bit fields corresponding to the DIO output enable states of each channel. |
| *pnStatus* | PSHORT | Returned status: 0 on success, negative number on failure. |

### Comments

Bit 0-7 of *pucOutputEnable* corresponds to DIO channels 0-7. A bit with a value of 1 represents an enabled state. A bit with a value of 0 represents a disabled state.

### Example

The following example gets the DIO output enable states:

```
SHORT nStatus;
BYTE  ucOutputEnable;

GxAoDioGetOutputEnable(nHandle, &ucOutputEnable, &nStatus);
if (ucOutputEnable & 1)
    printf("Channel 0 is enabled");
```

### See Also

**GxAoDioSetOutputEnable, GxAoGetErrorString**

## GxAoDioGetStatus

### Purpose

Returns the DIO sequencer status.

### Applies

**GX1649**

### Syntax

**GxAoDioGetStatus** (*nHandle, pdwStatus, pnStatus*)

### Parameters

| Name | Type | Comments |
|---|---|---|
| *nHandle* | SHORT | Handle to a GX1649 board. |
| *pdwStatus* | PDWORD | Returned the status of the DIO:<br>Bit 0: DIO Sequencer is Running<br>Bit 1: DIO Sequencer is Armed and ready for start trigger |
| *pnStatus* | PSHORT | Returned status: 0 on success, negative number on failure. |

### Example

The following example checks if the DIO is running

```
DWORD dwStatus;

GxAoDioGetStatus(nHandle, &dwStatus, &nStatus);
if ((dwStatus & 0x1)==1) printf("DIO is running");
```

### See Also

**GxDioArm, GxDioTrig, GxAoGetErrorString**

## GxAoDioGetVectorCount

### Purpose

Returns the number of vectors (steps) to run.

### Applies

**GX1649**

### Syntax

**GxAoDioGetVectorCount** (*nHandle, pdwVectorSize, pnStatus*)

### Parameters

| Name | Type | Comments |
|---|---|---|
| *nHandle* | SHORT | Handle to a GX1649 board. |
| *pdwVectorSize* | PDWORD | Returns the number of vectors to run (1-8192) |
| *pnStatus* | PSHORT | Returned status: 0 on success, negative number on failure. |

### Comments

This function returns the number of vectors that will be run during one iteration of a loop. Use the **GxAoDioArm** to setup the board to generate a loop or a single burst.

### Example

The following example gets the number of vectors to execute:

```
SHORT nStatus;
DWORD dwVectorSize;

GxAoDioGetVectorCount(nHandle, &dwVectorSize, &nStatus);
```

### See Also

**GxAoDioSetVectorCount, GxAoDioWriteMemory, GxAoGetErrorString**

## GxAoDioReadChannelMemory

### Purpose

Reads DIO direction, input or output memory per DIO channel

### Applies

**GX1649**

### Syntax

**GxAoDioReadChannelMemory** (*nHandle, nChannel, nMemoryType, pucVector, dwSize, pnStatus*)

### Parameters

| Name | Type | Comments |
|---|---|---|
| *nHandle* | SHORT | Handle to a GX1649 board. |
| *nChannel* | SHORT | DIO Channel (0-7) |
| *nMemoryType* | SHORT | Type of Memory to read from:<br>0. GX1649_DIO_MEMORY_INPUT_TYPE: Write to DIO Input Memory<br>1. GX1649_DIO_MEMORY_OUTPUT_TYPE: Write to DIO Output Memory<br>2. GX1649_DIO_MEMORY_DIRECTION_TYPE: Write DIO Direction Memory |
| *pucVector* | PBYTE | Buffer to hold the returned array of bytes that represent the memory states. Buffer size used is specified using the *dwSize* parameter. |
| *dwSize* | DWORD | The number of vectors to read from memory. |
| *pnStatus* | PSHORT | Returned status: 0 on success, negative number on failure. |

### Comments

Each element in the byte array represents the direction, input or output state of one DIO channel. A value of 0 or 1 is returned for each elements as specified by *nMemoryType*:

- GX1649_DIO_MEMORY_INPUT_TYPE: 1 represents a recorded logic high, and 0 represents a recorded logic low
- GX1649_DIO_MEMORY_OUTPUT_TYPE: 1 represents a logic high output, and 0 represents a logic low output
- GX1649_DIO_MEMORY_DIRECTION_TYPE: 1 represents output and a 0 represents input.

### Example

The following example reads 1000 vectors from the input output DIO memory of channel 5:

```
SHORT nStatus;
BYTE  aucVector[1000];

GxAoDioReadChannelMemory (nHandle, 5, GX1649_DIO_MEMORY_INPUT_TYPE, aucVector, 1000, &nStatus);
```

### See Also

**GxAoDioReadMemory, GxAoDioWriteChannelMemory, GxAoGetVectorCount, GxAoGetErrorString**

## GxAoDioReadMemory

### Purpose

Reads DIO direction, input or output memory using a byte array for all channels

### Applies

**GX1649**

### Syntax

**GxAoDioReadMemory** (*nHandle, nMemoryType, pucVector, dwSize, pnStatus*)

### Parameters

| Name | Type | Comments |
|---|---|---|
| *nHandle* | SHORT | Handle to a GX1649 board. |
| *nMemoryType* | SHORT | Type of Memory to read from:<br>0. GX1649_DIO_MEMORY_INPUT_TYPE: Write to DIO Input Memory<br>1. GX1649_DIO_MEMORY_OUTPUT_TYPE: Write to DIO Output Memory<br>2. GX1649_DIO_MEMORY_DIRECTION_TYPE: Write DIO Direction Memory |
| *pucVector* | PBYTE | Buffer to hold the returned array of bytes that represent the memory states. Buffer size is specified using the *dwSize* parameter. |
| *dwSize* | DWORD | The number of vectors to read from memory. Buffer size (*pucVector*) must be equal or longer than this value. |
| *pnStatus* | PSHORT | Returned status: 0 on success, negative number on failure. |

### Comments

Each element in the byte array represents the direction, input or output state of all 8 DIO channels. Each bit corresponds to a channel, bit 0 is channel 0, etc. The value of each bit is interpreted according to *nMemoryType*:

- GX1649_DIO_MEMORY_INPUT_TYPE: 1 represents a recorded logic high, and 0 represents a recorded logic low
- GX1649_DIO_MEMORY_OUTPUT_TYPE: 1 represents a logic high output, and 0 represents a logic low output
- GX1649_DIO_MEMORY_DIRECTION_TYPE: 1 represents output and a 0 represents input.

### Example

The following example reads 1000 vectors from the direction memory:

```
SHORT nStatus;
BYTE  aucVector[1000];

GxAoDioReadMemory (nHandle, GX1649_DIO_MEMORY_INPUT_TYPE, aucVector, 1000, &nStatus);
```

### See Also

**GxAoDioReadChannelMemory, GxAoDioWriteMemory, GxAoGetVectorCount, GxAoGetErrorString**

## GxAoDioSetClock

### Purpose

Sets the DIO clock source and frequency.

### Applies

**GX1649**

### Syntax

**GxAoDioSetClock** (*nHandle, nClockSource, dFrequency, pnStatus*)

### Parameters

| Name | Type | Comments |
|---|---|---|
| *nHandle* | SHORT | Handle to a GX1649 board. |
| *nClockSource* | SHORT | Sets clock source:<br>0. GXAO_1649_CLOCKSOURCE_INTERNAL<br>1. GXAO_1649_CLOCKSOURCE_EXTERNAL<br>2. GXAO_1649_CLOCKSOURCE_PXI0<br>3. GXAO_1649_CLOCKSOURCE_PXI1<br>4. GXAO_1649_CLOCKSOURCE_PXI2<br>5. GXAO_1649_CLOCKSOURCE_PXI3<br>6. GXAO_1649_CLOCKSOURCE_PXI4<br>7. GXAO_1649_CLOCKSOURCE_PXI5<br>8. GXAO_1649_CLOCKSOURCE_PXI6<br>9. GXAO_1649_CLOCKSOURCE_PXI7<br>10. GXAO_1649_CLOCKSOURCE_STAR<br>11. GXAO_1649_CLOCKSOURCE_GROUPA |
| *dFrequncy* | DOUBLE | Sets DIO frequency, this is only relevant when *pnClockSource* is control by the card internally (GXAO_1649_CLOCKSOURCE_INTERNAL) |
| *pnStatus* | PSHORT | Returned status: 0 on success, negative number on failure. |

### Comments

Default clock source is GXAO_1649_CLOCKSOURCE_INTERNAL with frequency of 10MHz
GXAO_1649_CLOCKSOURCE_GROUPA can only be used when *nGroup* is set to B, C, and D, using group A clock source will synchronizes all group channels to use the same clock source as group A.

### Example

The following example sets group A with clock source of internal with frequency of 5MHz:

```
SHORT nStatus;

GxAoDioSetClock (nHandle, GXAO_GROUPA, GXAO_1649_CLOCKSOURCE_INTERNAL, 5000000, &nStatus);
```

### See Also

**GxAoDioGetClock, GxAoGetErrorString**

## GxAoDioSetOutputEnable

### Purpose

Sets the Digital IO output enable states

### Applies

**GX1649**

### Syntax

**GxAoDioSetOutputEnable**(*nHandle*, *ucOutputEnable*, *pnStatus*)

### Parameters

| Name | Type | Comments |
|---|---|---|
| *nHandle* | SHORT | Handle to a GX1649 board. |
| *ucOutputEnable* | BYTE | Sets the bit field corresponding to the DIO output enable states. |
| *pnStatus* | PSHORT | Returned status: 0 on success, negative number on failure. |

### Comments

Bit 0-7 of ucOutputEnable correspond to DIO channels 0-7.

A bit with a value of 1 represents an enabled state. A bit with a value of 0 represents a disabled state.

### Example

The following example sets DIO Channel output enable states for DIO Channel 0 and 2 to Enabled:

```
SHORT nStatus;

GxAoDioSetOutputEnable(nHandle, 0x5, &nStatus);
```

### See Also

**GxAoDioGetOutputEnable, GxAoGetErrorString**

## GxAoDioSetVectorCount

### Purpose

Sets the number of vectors (steps) to run.

### Applies

**GX1649**

### Syntax

**GxAoDioSetVectorCount** (*nHandle, dwVectorSize, pnStatus*)

### Parameters

| Name | Type | Comments |
|---|---|---|
| *nHandle* | SHORT | Handle to a GX1649 board. |
| *dwVectorSize* | DWORD | Sets the number of vectors to run (1-8192) |
| *pnStatus* | PSHORT | Returned status: 0 on success, negative number on failure. |

### Comments

This function sets the number of vectors that will be run during one iteration of a loop. Use the **GxAoDioArm** to setup the board to generate a loop or a single burst.

### Example

The following example sets the number of vectors to 1000:

```
SHORT nStatus;

GxAoDioSetVectorCount(nHandle, 1000, &nStatus);
```

### See Also

**GxAoDioReadChannelMemory, GxAoDioWriteMemory, GxAoDioGetVectorCount, GxAoGetErrorString**

## GxAoDioTrig

### Purpose

Trigger the DIO and starts waveform generation.

### Applies

**GX1649**

### Syntax

**GxAoDioTrig** (*nHandle*, *pnStatus*)

### Parameters

| Name | Type | Comments |
|---|---|---|
| *nHandle* | SHORT | Handle to a GX1649 board. |
| *pnStatus* | PSHORT | Returned status: 0 on success, negative number on failure. |

### Comments

This function will start generating the waveform in all trigger modes. The DIO must be in armed state.

### Example

The following start the waveform generation:

```
SHORT nStatus;

GxAoDioArm (nHandle, TRUE, TRUE, &nStatus);
GxAoDioTrig (nHandle, &nStatus);
```

### See Also

**GxAoDioArm, GxAoGetErrorString**

## GxAoDioWriteChannelMemory

### Purpose

Writes DIO direction, input or output memory per DIO channel

### Applies

**GX1649**

### Syntax

**GxAoDioWriteChannelMemory** (*nHandle, nChannel, nMemoryType, pucVector, dwSize, pnStatus*)

### Parameters

| Name | Type | Comments |
|---|---|---|
| *nHandle* | SHORT | Handle to a GX1649 board. |
| *nChannel* | SHORT | DIO Channel (0-7). |
| *nMemoryType* | SHORT | Type of Memory to write to:<br>0. GX1649_DIO_MEMORY_INPUT_TYPE: Write to DIO Input Memory.<br>1. GX1649_DIO_MEMORY_OUTPUT_TYPE: Write to DIO Output Memory.<br>2. GX1649_DIO_MEMORY_DIRECTION_TYPE: Write DIO Direction Memory. |
| *pucVector* | PBYTE | Buffer to hold an array of bytes that represent the memory states. Buffer size used is specified using the *dwSize* parameter. |
| *dwSize* | DWORD | The number of vectors to write to memory. |
| *pnStatus* | PSHORT | Returned status: 0 on success, negative number on failure. |

### Comments

Each element in the byte array represents the direction, input or output state of one DIO channel. A value of 0 or 1 is returned for each elements as specified by *nMemoryType*:

- GX1649_DIO_MEMORY_INPUT_TYPE: 1 represents a recorded logic high, and 0 represents a recorded logic low
- GX1649_DIO_MEMORY_OUTPUT_TYPE: 1 represents a logic high output, and 0 represents a logic low output
- GX1649_DIO_MEMORY_DIRECTION_TYPE: 1 represents output and a 0 represents input.

### Example

The following example writes 1000 vectors to the output DIO memory of channel 5:

```
SHORT nStatus;
BYTE  aucVector[1000];

GxAoDioSetVectorCount(nHandle, 1000, &nStatus);
GxAoDioWriteChannelMemory (nHandle, 5, GX1649_DIO_MEMORY_OUTPUT_TYPE, aucVector, 1000, &nStatus);
```

### See Also

**GxAoDioReadChannelMemory, GxAoDioSetVectorCount, GxAoDioWriteMemory, GxAoGetErrorString**

## GxAoDioWriteMemory

### Purpose

Writes DIO direction, input or output memory using a byte array

### Applies

**GX1649**

### Syntax

**GxAoDioWriteMemory** (*nHandle, nMemoryType, pucVector, dwSize, pnStatus*)

### Parameters

| Name | Type | Comments |
|---|---|---|
| *nHandle* | SHORT | Handle to a GX1649 board. |
| *nMemoryType* | SHORT | Type of Memory to write to:<br>0. GX1649_DIO_MEMORY_INPUT_TYPE: Write to DIO Input Memory.<br>1. GX1649_DIO_MEMORY_OUTPUT_TYPE: Write to DIO Output Memory.<br>2. GX1649_DIO_MEMORY_DIRECTION_TYPE: Write DIO Direction Memory. |
| *pucVector* | PBYTE | Buffer to hold an array of bytes that represent the memory states. Buffer size used is specified using the *dwSize* parameter. |
| *dwSize* | DWORD | The number of vectors to write to memory. |
| *pnStatus* | PSHORT | Returned status: 0 on success, negative number on failure. |

### Comments

Each element in the byte array represents the direction, input or output state of all 8 DIO channels. Each bit corresponds to a channel, bit 0 is channel 0, etc. The value of each bit is interpreted according to *nMemoryType*:

- GX1649_DIO_MEMORY_INPUT_TYPE: 1 represents a recorded logic high, and 0 represents a recorded logic low
- GX1649_DIO_MEMORY_OUTPUT_TYPE: 1 represents a logic high output, and 0 represents a logic low output
- GX1649_DIO_MEMORY_DIRECTION_TYPE: 1 represents output and a 0 represents input.

### Example

The following example writes 1000 vectors to the DIO Direction memory:

```
SHORT nStatus;
BYTE  aucVector[1000];

GxAoDioSetVectorCount(nHandle, 1000, &nStatus);
GxAoDioWriteMemory (nHandle, GX1649_DIO_MEMORY_DIRECTION, aucVector, 1000, &nStatus);
```

### See Also

**GxAoDioReadMemory, GxAoDioSetVectorCount, GxAoDioWriteChannelMemory, GxAoGetErrorString**

## GxAoGetBoardAccuracy

### Purpose

Returns the board minimum per channel resolution.

### Syntax

**GxAoGetBoardAccuracy** (*nHandle, pnAccuracy, pnStatus*)

### Parameters

| Name | Type | Comments |
|---|---|---|
| *nHandle* | SHORT | Handle to a GX1642, GX1648 or GX1649 board. |
| *pnAccuracy* | PSHORT | Returns the board Accuracy, values are:<br>0. 2mV<br>1. 5mV |
| *pnStatus* | PSHORT | Returned status: 0 on success, negative number on failure. |

### Comments

Board accuracy depends on the board factory installed output amplifier type.

### Example

The following example returns the board accuracy:

```
SHORT nStatus, nAccuracy;

GxAoGetBoardAccuracy (nHandle, &nAccuracy, &nStatus);
```

### See Also

**GxAoGetBoardSummary, GxAoGetDriverSummary, GxAoGetErrorString**

## GxAoGetBoardSummary

### Purpose

Returns the board name, description, S/N, firmware version and revision, and calibration time.

### Syntax

**GxAoGetBoardSummary**(*nHandle, pszSummary, nSummaryMaxLen, pnStatus*)

### Parameters

| Name | Type | Comments |
|---|---|---|
| *nHandle* | SHORT | Handle to a GX1642, GX1648, or GX1649 board. |
| *pszSummary* | PSTR | Buffer to contain the returned board info (null terminated) string. |
| *nSummaryMax Len* | SHORT | Size of the buffer to contain the error string. |
| *pnStatus* | PSHORT | Returned status: 0 on success, negative number on failure. |

### Comments

The GXAO board summary string provides the following data from the on-board nonvolatile electrically erasable programmable read-only memory (EEPROM) in the order shown:

- Instrument Name (e.g., GX1648)
- Instrument Description (e.g. 48 Channels Analog Output)
- EEPROM format version (e.g., 1.00)
- PCB revision (e.g., 'A')
- FPGA version (e.g. 0xA003)
- Serial Number (e.g., 1648008)
- Calibration time and date

For example, the returned string could look like the following:

```
"GX1648 48 Channels Analog Outputs, EEPROM-Version:1, Board-Revision:A, FPGA-
   Version:0x0A03, S/N 1648002, Calibration-time:Wed Aug 14 17:30:04 2002"
```

### Example

The following example returns the board summary:

```
SHORT nHandle, nStatus;
CHAR szSummary[256];

GxAoGetBoardSummary(nHandle, szSummary, sizeof(szSummary), &nStatus)
```

### See Also

**GxAoGetDriverSummary, GxAoInitialize, GxAoGetErrorString**

## GxAoGetCalibrationInfo

### Purpose

Returns the calibration information.

### Syntax

**GxAoGetCalibrationInfo** (*nHandle, pszCalibrationInfo, nInfoMaxLen, pnDaysUntilExpire, pnStatus*)

### Parameters

| Name | Type | Comments |
|---|---|---|
| *nHandle* | SHORT | Handle to a GX1649 board |
| *pszSummary* | PSTR | Buffer to contain the returned board's calibration information (null terminated) string. |
| *nSumMaxLen* | SHORT | Size of the buffer to contain the error string. |
| *pnDaysUntilExpire* | PSHORT | Returns the number of days until or from expiration, if number is > 0 then calibration is current otherwise past due. |
| *pnStatus* | PSHORT | Returned status: 0 on success, negative number on failure. |

### Comments

The returned board's calibration information has the following fields:

Model: model number, e.g. "GX1649"

Serial Number: serial number, e.g. 216

Control Number: Marvin Test Solutions control number, e.g. "ZZ-AD-AA-000"

Production Calibration Date: Wed Oct 24 12:30:25 2010

Calibration Date: Wed Oct 24 12:31:58 2010

Recommended Interval: 1 year

Next Calibration Date: Fri Oct 24 12:31:58 2011

Status: calibration status can be either "Expired" followed by the number of days past expiration or "Current" followed by number of days until expire.

Calibration License: can be either "Installed" with the calibration license number or "Not Installed".

### Example

The following example returns the board's calibration information string:

```
SHORT   nStatus;
char    szCalibrationInfo[1024];
BOOL    bExpired;

GxAoGetCalibrationInfo(nHandle, szCalibrationInfo, sizeof szCalibrationInfo, &bExpired,
&nStatus);

szCalibrationInfo string printout:

Model: GX1649
Serial Number: 12
Control Number: ZZ-AD-AA-000
Production Calibration Date: Wed May 23 12:30:25 2012
Calibration Date: Wed Oct 24 12:31:58 2007
Recommended Interval: 1 year
Next Calibration Date: Fri May 24 12:31:58 2013
Status: Expired (891 days past expiration)
Calibration License: Installed license number 3
```

### See Also

**GxAoInitialize, GxAoGetBoardSummary**, **GxAoGetErrorString**

## GxAoGetChannelVoltage

### Purpose

Returns the specified channel's group voltage.

### Syntax

**GxAoGetChannelVoltage** (*nHandle, nGroup, nChannel, pdVoltage, pnStatus*)

### Parameters

| Name | Type | Comments |
|---|---|---|
| *nHandle* | SHORT | Handle to a GX1642, GX1648 or GX1649 board. |
| *nGroup* | SHORT | Group number:<br>0. GXAO_GROUPA<br>1. GXAO_GROUPB<br>2. GXAO_GROUPC<br>3. GXAO_GROUPD |
| *nChannel* | SHORT | Channel number 0-15. |
| *pdVoltage* | PDOUBLE | Returned channel voltage |
| *pnStatus* | PSHORT | Returned status: 0 on success, negative number on failure. |

### Example

The following example returns the voltage for group A channel 4:

```
SHORT nStatus;
DOUBLE dVoltage;

GxAoGetChannelVoltage (nHandle, GXAO_GROUPA, 4, &dVoltage, &nStatus);
```

### See Also

**GxAoSetChannelVoltage, GxAoSetGroupVoltageRange, GxAoGetGroupVoltageRange, GxAoGetErrorString**

## GxAoGetDriverSummary

### Purpose

Returns the driver name and version.

### Syntax

**GxAoGetDriverSummary** (*pszSummary ,nSummaryMaxLen, pdwVersion, pnStatus*)

### Parameters

| Name | Type | Comments |
|---|---|---|
| *pszSummary* | PSTR | Buffer to the returned driver summary string. |
| *nSummaryMaxLen* | SHORT | The size of the summary string buffer. |
| *pdwVersion* | PDWORD | Returned version number. The high order word specifies the major version number where the low order word specifies the minor version number. |
| *pnStatus* | PSHORT | Returned status: 0 on success, negative number on failure. |

### Comments

The returned string is:

```
"GXAO Driver for GX1648. Version 1.0, Copyright © Marvin Test Solutions 2013"
```

### Example

The following example prints the driver version:

```
CHAR    sz[128];
DWORD   dwVersion;
SHORT   nStatus;
GxAoGetDriverSummary (sz, sizeof sz, &dwVersion, &nStatus);
printf("Driver Version %d.%d", (INT)(dwVersion>>16), (INT)
        dwVersion &0xFFFF);
```

### See Also

**GxAoGetErrorString**

## GxAoGetErrorString

### Purpose

Returns the error string associated with the specified error number.

### Syntax

**GxAoGetErrorString** (*nError* , *pszMsg*, *nErrorMaxLen*, *pnStatus*)

### Parameters

| Name | Type | Comments |
|---|---|---|
| *nError* | SHORT | Error number. |
| *pszMsg* | PSTR | Buffer to the returned error string. |
| *nErrorMaxLen* | SHORT | The size of the error string buffer. |
| *pnStatus* | PSHORT | Returned status: 0 on success, negative number on failure. |

### Comments

The function returns the error string associated with the *nError* as returned from other driver functions.

The following table displays the possible error values; not all errors apply to this board type:

**Resource Errors**

| | |
|---|---|
| 0 | No error has occurred |
| -1 | Unable to open the HW driver. Check if HW is properly installed |
| -2 | Board does not exist in this slot/base address |
| -3 | Different board exist in the specified PCI slot/base address |
| -4 | PCI slot not configured properly. You may configure using the PciExplorer from the Windows Control Panel |
| -5 | Unable to register the PCI device |
| -6 | Unable to allocate system resource for the device |
| -7 | Unable to allocate memory |
| -8 | Unable to create panel |
| -9 | Unable to create Windows timer |
| -10 | Bad or wrong board EEPROM |
| -11 | Not in calibration mode |
| -12 | Board is not calibrated |
| -13 | Function is not supported by the specified board |

**General Parameter Errors**

| | |
|---|---|
| -20 | Invalid or unknown error number |
| -21 | Invalid parameter |
| -22 | Illegal slot number |
| -23 | Illegal board handle |

-24 Illegal string length

-25 Illegal operation mode

**Parameter Errors**

-50 Invalid group number, allowed range is 0-3

-51 Invalid channel number, allowed range is 0-15

-52 Invalid range voltage range, allowed range is 0-3

-53 Unable to set I/O port since port direction is set to input

-54 Invalid port bit number, allowed range is 0-3

-55 Voltage is out of the allowed range

-56 Channel is busy, return on timeout

-57 Unable to lock channel or group for voltage settings

-58 Invalid clock rate, allowed range is 0.5-10000000

-59 ARB memory channels have changed, memory is out of date

-60 Channel is not configured as ARB

-61 ARB memory size exceeded, allowed range is 1-262144

-62 Channels is configured as Static, not ARB

-63 Cannot continue operation because the ARB sequencer is running

-64 Invalid calibration Channel Gain calculated

-65 Invalid calibration ADC Gain calculated

-66 Invalid calibration ADC Offset calculated

-67 Invalid trigger mode, allowed range is 0-11

-68 Invalid clock source, allowed range is 0-11

-69 DIO memory size exceeded, allowed range is 1-8192

-70 Invalid DIO memory type, allowed range is 0-2

-71 Allocated DIO vector size exceeded

-72 Calibration mode is not enabled

-73 Invalid calibration voltage point

-74 Cannot continue operation because the DIO is running

### Example

The following example initializes the board. If the initialization failed, the following error string is printed:

```
CHAR    sz[256];
SHORT   nStatus, nHandle;

GxAoInitialize (3, &Handle, &Status);
if (nStatus<0)
{   GxAoGetErrorString(nStatus, sz, sizeof sz, &nStatus);
    printf(sz); // prints the error string returns
}
```

## GxAoGetGroupExternalUpdate

### Purpose

Returns the specified group external update enable state.

### Syntax

**GxAoGetGroupExternalUpdate** (*nHandle, nGroup, pbEnabled, pnStatus*)

### Parameters

| Name | Type | Comments |
|---|---|---|
| *nHandle* | SHORT | Handle to a GX1642 GX1648 or GX1649 board. |
| *nGroup* | SHORT | Group number:<br>0. GXAO_GROUPA<br>1. GXAO_GROUPB<br>2. GXAO_GROUPC<br>3. GXAO_GROUPD |
| *pbEnabled* | PBOOL | Return TRUE if the group external update is enabled and FALSE if disabled. |
| *pnStatus* | PSHORT | Returned status: 0 on success, negative number on failure. |

### Comments

When enabled a low on the group's external update pin will continuously update the group voltage causing its DACs to be loaded from the channel register as set by the **GxAoSetChannelVoltage**.

### Example

The following example returns group A external update enable state:

```
SHORT nStatus
BOOL bEnabled;

GxAoGetGroupExternalUpdate (nHandle, GXAO_GROUPA, &bEnabled, &nStatus);
```

### See Also

**GxAoSetGroupExternalUpdate, GxAoSetChannelVoltage, GxAoSetGroupVoltageRange, GxAoGetGroupVoltageRange, GxAoGetErrorString**

## GxAoGetGroupVoltageRange

### Purpose

Returns the specified group voltage range.

### Applied To

**GX1642**, **GX1648**

### Syntax

**GxAoGetGroupVoltageRange** (*nHandle, nGroup, pnVoltageRange, pnStatus*)

### Parameters

| Name | Type | Comments |
|---|---|---|
| *nHandle* | SHORT | Handle to a GX1642 or GX1648 board. |
| *nGroup* | SHORT | Group number:<br>0. GXAO_GROUPA<br>1. GXAO_GROUPB<br>2. GXAO_GROUPC<br>3. GXAO_GROUPD |
| *pnVoltageRange* | PSHORT | Return the specified group voltage range.<br>**Gx1642:**<br>0. GXAO_1642_ZERO_TO_POS10V: 0 to 20V (default)<br>1. GXAO_1642_NEG10V_TO_POS10V: -20 to 20V<br>2. GXAO_1642_ZERO_TO_POS5V: 0 to 10V<br>3. GXAO_1642_NEG5V_TO_POS5V: -10 to -10V<br><br>**Gx1648:**<br>0. GXAO_ZERO_TO_POS10V: 0 to 10V (default)<br>1. GXAO_NEG10V_TO_POS10V: -10 to 10V<br>2. GXAO_ZERO_TO_POS5V: 0 to 5V<br>3. GXAO_NEG5V_TO_POS5V: -5 to -5V |
| *pnStatus* | PSHORT | Returned status: 0 on success, negative number on failure. |

### Comments

The voltage resolutions of all the channels in the specified group will increase/decrease as follow:

Gx1642:

| **Range** | **Resolution** |
|---|---|
| 0 to 20V | 4.88mV |
| -20 to 20V | 9.76 mV |
| 0 to 10V | 2.44 mV |
| -10 to –10V | 4.88mV |

Gx1648:

| **Range** | **Resolution** |
|---|---|
| 0 to 10V | 2.44mV |
| -10 to 10V | 4.88 mV |
| 0 to 5V | 1.22 mV |
| -5 to –5V | 2.44mV |

Any call to **GxAoSetGroupVoltageRange** resets all groups ’channels to zero volts.

### Example

The following example returns group A voltage range:

```
SHORT nStatus, nVoltageRange;

GxAoGetGroupVoltageRange (nHandle, GXAO_GROUPA, &nVoltageRange, &nStatus);
```

### See Also

**GxAoSetGroupVoltageRange, GxAoLoadChannelVoltage, GxAoUpdateGroupVoltage, GxAoGetErrorString**

## GxAoGetGroupUpdateState

### Purpose

Returns whether the group channels loaded values were loaded to it DACs.

### Syntax

**GxAoGetGroupUpdateState** (*nHandle, nGroup, pbUpdated, pnStatus*)

### Parameters

| Name | Type | Comments |
|---|---|---|
| *nHandle* | SHORT | Handle to a GX1642, GX1648 or GX1649 board. |
| *nGroup* | SHORT | Group number:<br>0. GXAO_GROUPA<br>1. GXAO_GROUPB<br>2. GXAO_GROUPC<br>3. GXAO_GROUPD |
| *pbUpdated* | PBOOL | Returns low if the specified group was not updated, high if it was updated |
| *pnStatus* | PSHORT | Returned status: 0 on success, negative number on failure. |

### Comments

Loading new value to any or all the specified group channels using GxAo**SetChannelVoltage** will set *pbUpdated* to FALSE. Updating the specified group channels will set load the value to the board DAC and will cause *pbUpdated* to return TRUE.

### Example

The following example returns group A external update enable state:

```
SHORT nStatus;
BOOL pbnUpdated;

GxAoGetGroupUpdateState (nHandle, GXAO_GROUPA, &bUpdated, &nStatus);
```

### See Also

**GxAoGetGroupExternalUpdate**, **GxAoSetChannelVoltage, GxAoLoadChannelVoltage, GxAoUpdateGroupVoltage, GxAoGetErrorString**

## GxAoGetPortBit

### Purpose

Returns the digital output bit state.

### Applied To

**GX1642**, **GX1648**

### Syntax

**GxAoGetPortBit** (*nHandle, pbHi, pnStatus*)

### Parameters

| Name | Type | Comments |
|---|---|---|
| *nHandle* | SHORT | Handle to a GX1642 or GX1648 board. |
| *pbHi* | PBOOL | Digital out bit line state: TRUE for high, FALSE for low. |
| *pnStatus* | PSHORT | Returned status: 0 on success, negative number on failure. |

### Example

The following example returns the Digital I/O line state:

```
SHORT nStatus;
BOOL  bHi;

GxAoGetPortBit (nHandle, &bHi, &nStatus);
```

### See Also

**GxAoSetPortBit, GxAoGetErrorString**

## GxAoInitialize

### Purpose

Initializes the driver for the specified PXI slot using the HW device driver.

### Syntax

**GxAoInitialize** (*nSlot, pnHandle, pnStatus*)

### Parameters

| Name | Type | Comments |
|---|---|---|
| *nSlot* | SHORT | Counter board slot number. See Comments. |
| *pnHandle* | PSHORT | Returned Handle for a Counter board. |
| *pnStatus* | PSHORT | Returned status: 0 on success, 1 on failure. |

### Comments

The Marvin Test Solutions HW device driver is installed with the driver and is the default device driver. The function returns a handle that for use with other Counter functions to program the board. The function does not change any of the board settings.

The specified PXI slot number is displayed by the **PXI/PCI Explorer** applet that can be opened from the Windows **Control Panel**. You may also use the label on the chassis below the PXI slot where the board is installed. The function accepts two types of slot numbers:

- A combination of chassis number (chassis # x 256) with the chassis slot number. For example 0x105 (chassis 1 slot 5).
- Legacy nSlot as used by earlier versions of HW/VISA. The slot number contains no chassis number and can be changed using the **PXI/PCI Explorer** applet (1-255).

### Example

The following example initializes the board at PXI chassis 2 slot 7.

```
SHORT nHandle, nStatus;
GxAoInitialize(0x207, &nHandle, &nStatus);
if (nHandle==0)
{
    printf("Unable to Initialize the board")
    return;
}
```

### See Also

**GxAoInitalizeVisa, GxAoGetErrorString, GxAoReset**

## GxAoInitializeVisa

### Purpose

Initializes the driver for the specified PXI slot using the default VISA provider.

### Syntax

**GxAoInitializeVisa** (*szVisaResource, pnHandle, pnStatus*)

### Parameters

| Name | Type | Comments |
|---|---|---|
| *szVisaResource* | LPCTSTR | String identifying the location of the specified board in order to establish a session. |
| *pnHandle* | PSHORT | Returned Handle (session identifier) that can be used to call any other operations of that resource |
| *pnStatus* | PSHORT | Returned status: 0 on success, 1 on failure. |

### Comments

The **GxAoInitializeVisa** opens a VISA session to the specified resource. The function uses the default VISA provider configured in your system to access the board. You must ensure that the default VISA provider support PXI/PCI devices and that the board is visible in the VISA resource manager before calling this function.

The first argument *szVisaResource* is a string that is displayed by the VISA resource manager such as NI Measurement and Automation (NI_MAX). It is also displayed by Marvin Test Solutions PXI/PCI Explorer as shown in the prior figure. The VISA resource string can be specified in several ways as follows:

- Using chassis, slot: “PXI0::CHASSIS1::SLOT5”
- Using the PCI Bus/Device combination: “PXI9::13::INSTR” (bus 9, device 9).
- Using alias: “COUNTER1”. Use the PXI/PCI Explorer to set the device alias.

The function returns a board handle (session identifier) that can be used to call any other operations of that resource. The session is opened with VI_TMO_IMMEDIATE and VI_NO_LOCK VISA attributes. On terminating the application the driver automatically invokes **viClose**() terminating the session.

### Example

The following example initializes a Counter boards at PXI bus 5 and device 11.

```
SHORT nHandle, nStatus;
GxAoInitializeVisa(“PXI5::11::INSTR”, &nHandle, &nStatus);
if (nHandle==0)
{
    printf("Unable to Initialize the board")
    return;
}
```

### See Also

**GxAoInitialize, GxAoGetErrorString, GxAoReset**

## GxAoLoadChannelVoltage

### Purpose

Load the specified groups' channel with new voltage value.

### Syntax

**GxAoLoadChannelVoltage** (*nHandle, nGroup, nChannel, dVoltage, pnStatus*)

### Parameters

| Name | Type | Comments |
|---|---|---|
| *nHandle* | SHORT | Handle to a GX1642, GX1648 or GX1649 board. |
| *nGroup* | SHORT | Group number:<br>0. GXAO_GROUPA<br>1. GXAO_GROUPB<br>2. GXAO_GROUPC<br>3. GXAO_GROUPD |
| *nChannel* | SHORT | Channel number 0-15. |
| *dVoltage* | DOUBLE | Channel new voltage value. |
| *pnStatus* | PSHORT | Returned status: 0 on success, negative number on failure. |

### Comments

Loading a channel with a new value does not affect the channels output. The channel's DAC will be programmed with the new value only after calling **GxAoUpdateGroupVoltage**. Calling **GxAoSetChannelVoltage** will only update all the channels in the specified group with their current loaded values. This function allows the user to load different values to different channels/groups and update all the channels at once.

Calling this function will set **GxAoGetGroupUpdateState** to high (i.e. new value is waiting for the group to be updated).

If the new voltage is out of range the function will return an error.

### Example

The following example load channel five group B with 5.8 volts:

```
SHORT nStatus, nDirection;
GxAoLoadChannelVoltage (nHandle, GXAO_GROUPB, 5, 5.8, &nStatus);
```

### See Also

**GxAoSetChannelVoltage, GxAoGetChannelVoltage, GxAoGetGroupUpdateState, GxAoUpdateGroupVoltage, GxAoGetErrorString**

## GxAoPanel

### Purpose

Opens a virtual panel used to interactively control the GX1642/GX1648/GX1649.

### Syntax

**GxAoPanel** (*pnHandle, hwndParent, nMode, phwndPanel, pnStatus*)

### Parameters

| Name | Type | Comments |
|---|---|---|
| *pnHandle* | PSHORT | Handle to a GX1642, GX1648 or GX1649 board. |
| *hwndParent* | HWND | Panel parent window handle. A value of 0 sets the desktop as the parent window. |
| *NMode* | SHORT | The mode in which the panel main window is created. 0 for modeless window and 1 for modal window. |
| *phwndPanel* | PHWND | Returned window handle for the panel. |
| *pnStatus* | PSHORT | Returned status: 0 on success, negative number on failure. |

### Comments

The function is used to create the panel window. The panel window may be opened as a modal or a modeless window depending on the *nMode* parameters.

If the mode is set to modal dialog (*nMode*=1), the panel will disable the parent window (*hwndParent*) and the function will return only after the window was closed by the user. In that case, the *pnHandle* may return the handle created by the user using the panel Initialize dialog. This handle may be used when calling other GX1648 functions.

If a modeless dialog was created (*nMode*=0), the function returns immediately after creating the panel window returning the window handle to the panel - *phwndPanel.* It is the responsibility of the calling program to dispatch windows messages to this window so that the window can respond to messages.

### Example

The following example opens the panel in modal mode:

```
DWORD dwPanel;
SHORT nHandle=0, nStatus;
GxAoPanel(&nHandle, 0, 1, &dwPanel, &nStatus);
```

### See Also

**GxAoInitialize, GxAoGetErrorString**

## GxAoReset

### Purpose

Resets the GX1642/GX1648/GX1649 board to its default settings.

### Syntax

**GxAoReset** (*nHandle*, *pnStatus*)

### Parameters

| Name | Type | Comments |
|---|---|---|
| *nHandle* | SHORT | Handle to a GX1642, GX1648 or GX1649 board. |
| *pnStatus* | PSHORT | Returned status: 0 on success, negative number on failure. |

### Comments

The function does the following:

- All groups' voltage ranges are set to 0-10V.
- All channels of all groups are set to 0V.
- All external updates are disabled
- TTL out bit values are set to zero
- All Channels set to Static mode (GX1649 Only)
- All DIO Channels output disabled (GX1649 Only)
- ARB memory size set to 1 (GX1649 Only)
- ARB sample clock set to 100000 Hz
- DIO memory size set to 1 (GX1649 Only)
- DIO sample clock set to 100000 Hz

### Example

The following example initializes and resets the GX164X board:

```
GxAoInitialize (1, &nHandle, &nStatus);
GxAoReset (nHandle, &nStatus);
```

### See Also

**GxAoInitialize, GxAoGetErrorString**

## GxAoResetGroup

### Purpose

Resets the specified group to the default state.

### Syntax

**GxAoResetGroup** (*nHandle, nGroup, pnStatus*)

### Parameters

| Name | Type | Comments |
|---|---|---|
| *nHandle* | SHORT | Handle to a GX1642, GX1648 or GX1649 board. |
| *nGroup* | SHORT | Group number:<br>0. GXAO_GROUPA<br>1. GXAO_GROUPB<br>2. GXAO_GROUPC<br>3. GXAO_GROUPD |
| *pnStatus* | PSHORT | Returned status: 0 on success, negative number on failure. |

### Comments

The specified groups will be reset to:

- Group's channels set to 0V.
- Group's voltage range is set to 0-20V (GX1642) or 0-10 (GX1648).
- External update is disabled

### Example

The following example resets group A:

```
SHORT nStatus;

GxAoResetGroup(nHandle, GXAO_GROUPA, &nStatus);
```

### See Also

**GxAoSetChannelVoltage, GxAoGetChannelVoltage, GxAoSetGroupVoltageRange, GxAoGetGroupVoltageRange, GxAoGetErrorString**

## GxAoSelfTest

### Purpose

Performs a self-test to validate the card functionality and accuracy.

### Syntax

**GxAoSelfTest** (*nHandle,padwTestResult, pszTestResult, pnStatus*)

### Parameters

| Name | Type | Comments |
|---|---|---|
| *nHandle* | SHORT | Handle to a GX1649 board. |
| *padwTestResult* | PDWORD | Returns a 72 element array that contains the result of the self-test |
| *pszTestResult* | PSTR | Returns a string that summarizes the result of the self-test. |
| *pnStatus* | PSHORT | Returned status: 0 on success, negative number on failure. |

### Comments

The self-test performs the following tests:

- Output voltage test for each of the 64 channels (4 Groups of 16 Channels) and verification of accuracy within 5mV
- ARB memory test for each Group (Group A to Group D).
- DIO input, output, and direction memory tests
- Dynamic DIO test (internal loopback)

The test result is returned in a 72 element DWORD array. Each element is an individual test result. Elements 0 to 63 are the test results from the channel voltage output tests, element 64 to 67 are the ARB Group A to Group D memory tests, element 68 is the DIO Input Memory test, element 69 is the DIO Output Memory test, element 70 is the DIO Direction memory test and element 71 is the DIO Dynamic test.

| Element | Test Type | Results |
|---|---|---|
| 0 to 15 | Group A Channels 0-15 Output Tests | Pass=0, Fail=1 |
| 16 to 31 | Group B Channels 0-15 Output Tests | Pass=0, Fail=1 |
| 32 to 47 | Group C Channels 0-15 Output Tests | Pass=0, Fail=1 |
| 48 to 63 | Group D Channels 0-15 Output Tests | Pass=0, Fail=1 |
| 64 | ARB Group A Memory Test | Pass=0, Fail=1 |
| 65 | ARB Group B Memory Test | Pass=0, Fail=1 |
| 66 | ARB Group C Memory Test | Pass=0, Fail=1 |
| 67 | ARB Group D Memory Test | Pass=0, Fail=1 |
| 68 | DIO Input Memory Test | Pass=0, Fail=1 |
| 69 | DIO Output Memory Test | Pass=0, Fail=1 |
| 70 | DIO Direction Memory Test | Pass=0, Fail=1 |
| 71 | DIO Dyanmic IO Test | Pass=0, Fail=1 |

Ensure that the J6 connector is not connected before starting the self-test.

### Example

The following example performs a self-test:

```
SHORT nStatus;
DWORD adwTestResult[72];
CHAR  szTestResult[512];

GxAoSelfTest (nHandle, adwTestResult, szTestResult &nStatus);
if(adwTestResults[17]!=0)
                      printf("Group B, Channel 1 output voltage test failure");
```

### See Also

**GxAoReset, GxAoGetErrorString**

## GxAoSetChannelVoltage

### Purpose

Sets the specified channel's group voltage.

### Syntax

**GxAoSetChannelVoltage** (*nHandle, nGroup, nChannel, dVoltage, pnStatus*)

### Parameters

| Name | Type | Comments |
|---|---|---|
| *nHandle* | SHORT | Handle to a GX1642, GX1648 or GX1649 board. |
| *nGroup* | SHORT | Group number:<br>0. GXAO_GROUPA<br>1. GXAO_GROUPB<br>2. GXAO_GROUPC<br>3. GXAO_GROUPD |
| *nChannel* | SHORT | Channel number 0-15. |
| *dVoltage* | DOUBLE | Channel's voltage |
| *pnStatus* | PSHORT | Returned status: 0 on success, negative number on failure. |

### Comments

The voltage must be in the group's voltage range otherwise the function returns an error.

Calling this function will set **GxAoGetGroupUpdateState** to low (i.e. the group was updated).

### Example

The following example sets the voltage for group A channel 4 to 2.34V:

```
SHORT nStatus;

GxAoGetChannelVoltage (nHandle, GXAO_GROUPA, 4, 2.34, &nStatus);
```

### See Also

**GxAoGetChannelVoltage, GxAoSetGroupVoltageRange, GxAoGetGroupVoltageRange, GxAoGetErrorString**

## GxAoSetGroupExternalUpdate

### Purpose

Enables the specified group external update.

### Syntax

**GxAoSetGroupExternalUpdate** (*nHandle, nGroup, bEnabled, pnStatus*)

### Parameters

| Name | Type | Comments |
|---|---|---|
| *nHandle* | SHORT | Handle to a GX1642, GX1648 or GX1649 board. |
| *nGroup* | SHORT | Group number:<br>0. GXAO_GROUPA<br>1. GXAO_GROUPB<br>2. GXAO_GROUPC<br>3. GXAO_GROUPD |
| *bEnabled* | BOOL | A high enable the group external update line, a low disable the external update line. |
| *pnStatus* | PSHORT | Returned status: 0 on success, negative number on failure. |

### Comments

When enabled a low on the group's external update pin will continuously update the group voltage.

### Example

The following example enable group A external update:

```
SHORT nStatus;

GxAoSetChannelVoltage (nHandle, GXAO_GROUPA, TRUE, &nStatus);
```

### See Also

**GxAoGetGroupExternalUpdate, GxAoSetChannelVoltage, GxAoSetGroupVoltageRange, GxAoGetGroupVoltageRange, GxAoGetErrorString**

## GxAoSetGroupVoltageRange

### Purpose

Sets the specified group voltage range.

### Applied To

**GX1642**, **GX1648**

### Syntax

**GxAoSetGroupVoltageRange** (*nHandle, nGroup, nVoltageRange, pnStatus*)

### Parameters

| Name | Type | Comments |
|---|---|---|
| *nHandle* | SHORT | Handle to a GX1642 or GX1648 board. |
| *nGroup* | SHORT | Group number:<br>0. GXAO_GROUPA<br>1. GXAO_GROUPB<br>2. GXAO_GROUPC<br>3. GXAO_GROUPD |
| *nVoltageRange* | SHORT | Set the specified group voltage range for GX1642:<br>0. 0 to 20V (default)<br>1. -20 to 20V<br>2. 0 to 10V<br>3. -10 to -10V<br>For GX1648:<br>0. 0 to 10V (default)<br>1. -10 to 10V<br>2. 0 to 5V<br>3. -5 to -5V |
| *pnStatus* | PSHORT | Returned status: 0 on success, negative number on failure. |

### Comments

The voltage resolutions of all the channels in the specified group will increase/decrease as follow:

**Gx1642:**

| **Range** | **Resolution** |
|---|---|
| 0 to 20V | 4.88mV |
| -20 to 20V | 9.76 mV |
| 0 to 10V | 2.44 mV |
| -10 to –10V | 4.88mV |

**Gx1648:**

| **Range** | **Resolution** |
|---|---|
| 0 to 10V | 2.44mV |
| -10 to 10V | 4.88 mV |
| 0 to 5V | 1.22 mV |
| -5 to –5V | 2.44mV |

Any call to "**GxAoSetGroupVoltageRange**" resets all groups 'channels to zero volts.

### Example

The following example set group A voltage range to –5 to +5V:

```
SHORT nStatus;

GxAoSetGroupVoltageRange (nHandle, GXAO_GROUPA, 3, &nStatus);
```

### See Also

**GxAoGetGroupVoltageRange, GxAoLoadChannelVoltage, GxAoUpdateGroupVoltage, GxAoGetErrorString**

## GxAoSetPortBit

### Purpose

Sets the digital output bit state.

### Applied To

**GX1642**, **GX1648**

### Syntax

**GxAoSetPortBit** (*nHandle*, *bHi*, *pnStatus*)

### Parameters

| Name | Type | Comments |
|---|---|---|
| *nHandle* | SHORT | Handle to a GX1642 or GX1648 board. |
| *bHi* | BOOL | Digital out bit state one for high, 0 for low. |
| *pnStatus* | PSHORT | Returned status: 0 on success, negative number on failure. |

### Example

The following example sets Digital Output line high:

```
SHORT nStatus;

GxAoSetPortBit (nHandle, TRUE, &nStatus);
```

### See Also

**GxAoGetPortBit, GxAoGetErrorString**

## GxAoUpdateAllGroupsVoltage

### Purpose

Update all groups' channels outputs with the latest load values.

### Syntax

**GxAoUpdateAllGroupsVoltage** (*nHandle, pnStatus*)

### Parameters

| Name | Type | Comments |
|---|---|---|
| *nHandle* | SHORT | Handle to a GX1642, GX1648 or GX1649 board. |
| *pnStatus* | PSHORT | Returned status: 0 on success, negative number on failure. |

### Example

The following example updates all the groups:

```
SHORT nStatus;

GxAoUpdateAllGroupsVoltage (nHandle, &nStatus);
```

### See Also

**GxAoUpdateAllGroupsVoltage**, **GxAoSetGroupVoltageRange**, **GxAoGetGroupVoltageRange**, **GxAoLoadChannelVoltage**, **GxAoGetErrorString**

## GxAoUpdateGroupVoltage

### Purpose

Update all the specified group channels outputs with the load latest values.

### Syntax

**GxAoUpdateGroupVoltage** (*nHandle, nGroup, pnStatus*)

### Parameters

| Name | Type | Comments |
|---|---|---|
| *nHandle* | SHORT | Handle to a GX1642, GX1648 or GX1649 board. |
| *nGroup* | SHORT | Group number:<br>0. GXAO_GROUPA<br>1. GXAO_GROUPB<br>2. GXAO_GROUPC<br>3. GXAO_GROUPD |
| *pnStatus* | PSHORT | Returned status: 0 on success, negative number on failure. |

### Comments

The function updates all the specified group channels outputs with the currently loaded values.

### Example

The following example updates group B:

```
SHORT nStatus;

GxAoUpdateGroupVoltage (nHandle, GXAO_GROUPB, &nStatus);
```

### See Also

**GxAoSetGroupVoltageRange, GxAoGetGroupVoltageRange, GxAoLoadChannelVoltage, GxAoGetErrorString**

# Index

**H**



**I**



**J**



**L**



**M**



**N**



**O**



**P**

**R**



**S**



**T**



**U**



**V**