

16AIO

**16-Bit ADC/DAC, 32 Scanned Analog Inputs
4 Analog Outputs, 16-bit Digital I/O**

**All Form Factors
All 16AIO/LCAIO Models
All 12AIO/LCAIO Models**

API Library Reference Manual

**Manual Revision: March 17, 2026
Driver Release Version 6.0.117.x.x**

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

Preface

Copyright © 2018-2026, **General Standards Corporation**

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

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

General Standards Corporation makes no warranty of any kind with regard to this documentation and/or software, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. Although extensive editing and reviews are performed before release, **General Standards Corporation** assumes no responsibility for any errors, inaccuracies or omissions herein. This documentation, information and software are made available solely on an “as-is” basis. Nor is there any commitment to update or keep current this documentation.

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

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

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

ALL RIGHTS RESERVED.

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

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

TenAsys® and INtime® are registered trademarks of TenAsys.

Table of Contents

1. Introduction.....	7
1.1. Purpose.....	7
1.2. Acronyms.....	7
1.3. Definitions	7
1.4. Software Overview	7
1.4.1. Basic Software Architecture	7
1.4.2. API Library.....	8
1.4.3. Device Driver	8
1.5. Hardware Overview	8
1.6. Reference Material.....	9
1.7. Licensing.....	9
1.8. API Naming Conventions, Asynchronous I/O and glibc 2.34	9
2. Installation	10
2.1. Host and Environment Support.....	10
2.2. Driver and Device Information	10
2.3. File List.....	10
2.4. Directory Structure.....	10
2.5. Installation	11
2.6. Removal.....	11
2.7. Overall Make Script.....	11
2.8. Environment Variables	11
3. Main Interface Files.....	12
3.1. Main Header File	12
3.2. Main Library File.....	12
3.2.1. Build	12
3.2.2. Additional Libraries.....	12
4. API Library	13
4.1. Files.....	13
4.2. Build	13
4.3. Library Use	13
4.4. Macros	13
4.4.1. IOCTL Services.....	13
4.4.2. Registers	13
4.5. Data Types	14
4.6. Functions.....	14
4.6.1. gs_16aio_close().....	14
4.6.2. gs_16aio_init().....	15

4.6.3. gs_16aio_ioctl()	16
4.6.4. gs_16aio_open()	16
4.6.5. gs_16aio_read()	18
4.6.6. gs_16aio_write()	18
4.7. IOCTL Services	19
4.7.1. GS_16AIO_IOCTL_AI_BUF_CLEAR	19
4.7.2. GS_16AIO_IOCTL_AI_BUF_THR_LVL	20
4.7.3. GS_16AIO_IOCTL_AI_BUF_THR_STS	20
4.7.4. GS_16AIO_IOCTL_AI_MODE	20
4.7.5. GS_16AIO_IOCTL_AI_SCAN_1_CHAN	21
4.7.6. GS_16AIO_IOCTL_AI_SCAN_CLK_SRC	21
4.7.7. GS_16AIO_IOCTL_AI_SCAN_SIZE	21
4.7.8. GS_16AIO_IOCTL_AI_SYNC	22
4.7.9. GS_16AIO_IOCTL_AO_BUF_CLEAR	22
4.7.10. GS_16AIO_IOCTL_AO_BUF_THR_LVL	22
4.7.11. GS_16AIO_IOCTL_AO_BUF_THR_STS	22
4.7.12. GS_16AIO_IOCTL_AO_BURST_CLK_SRC	23
4.7.13. GS_16AIO_IOCTL_AO_BURST_ENABLE	23
4.7.14. GS_16AIO_IOCTL_AO_CLK_SRC	23
4.7.15. GS_16AIO_IOCTL_AO_LOOPING	24
4.7.16. GS_16AIO_IOCTL_AO_SYNC	24
4.7.17. GS_16AIO_IOCTL_AO_TIMING	24
4.7.18. GS_16AIO_IOCTL_AUTOCAL	25
4.7.19. GS_16AIO_IOCTL_AUTOCAL_STATUS	25
4.7.20. GS_16AIO_IOCTL_AUX_READ	25
4.7.21. GS_16AIO_IOCTL_AUX_WRITE	25
4.7.22. GS_16AIO_IOCTL_DATA_FORMAT	26
4.7.23. GS_16AIO_IOCTL_DIO_DIR_OUT	26
4.7.24. GS_16AIO_IOCTL_DIO_READ	26
4.7.25. GS_16AIO_IOCTL_DIO_WRITE	27
4.7.26. GS_16AIO_IOCTL_EXT_SYNC_SRC	27
4.7.27. GS_16AIO_IOCTL_INITIALIZE	27
4.7.28. GS_16AIO_IOCTL_IRQ0_SEL	27
4.7.29. GS_16AIO_IOCTL_IRQ1_SEL	28
4.7.30. GS_16AIO_IOCTL_IRQ2_SEL	28
4.7.31. GS_16AIO_IOCTL_QUERY	29
4.7.32. GS_16AIO_IOCTL_RANGE	30
4.7.33. GS_16AIO_IOCTL_REG_MOD	30
4.7.34. GS_16AIO_IOCTL_REG_READ	31
4.7.35. GS_16AIO_IOCTL_REG_WRITE	31
4.7.36. GS_16AIO_IOCTL_RGA_ENABLE	32
4.7.37. GS_16AIO_IOCTL_RGA_NRATE	32
4.7.38. GS_16AIO_IOCTL_RGB_CLK_SRC	32
4.7.39. GS_16AIO_IOCTL_RGB_ENABLE	33
4.7.40. GS_16AIO_IOCTL_RGB_NRATE	33
4.7.41. GS_16AIO_IOCTL_RX_IO_ABORT	33
4.7.42. GS_16AIO_IOCTL_RX_IO_MODE	33
4.7.43. GS_16AIO_IOCTL_RX_IO_TIMEOUT	34
4.7.44. GS_16AIO_IOCTL_TX_IO_ABORT	34
4.7.45. GS_16AIO_IOCTL_TX_IO_MODE	34
4.7.46. GS_16AIO_IOCTL_TX_IO_TIMEOUT	35
4.7.47. GS_16AIO_IOCTL_WAIT_CANCEL	35
4.7.48. GS_16AIO_IOCTL_WAIT_EVENT	36
4.7.49. GS_16AIO_IOCTL_WAIT_STATUS	38

5. The Driver.....	39
5.1. Files.....	39
5.2. Build	39
5.3. Startup.....	39
5.4. Verification	39
5.5. Version.....	39
5.6. Shutdown	39
6. Document Source Code Examples.....	40
6.1. Files.....	40
6.2. Build	40
6.3. Library Use	40
7. Utilities Source Code.....	41
7.1. Files.....	41
7.2. Build	41
7.3. Library Use	41
8. Operating Information	42
8.1. Debugging Aids	42
8.1.1. Device Identification	42
8.1.2. Detailed Register Dump	42
8.2. Basic Analog Input Configuration	42
8.3. Basic Analog Output Configuration.....	43
8.4. Data Transfer Modes.....	43
8.4.1. PIO - Programmed I/O	43
8.4.2. DMA - Block Mode DMA	43
9. Sample Applications	44
Document History	45

Table of Figures

Figure 1 Basic architectural representation.....8

1. Introduction

1.1. Purpose

The purpose of this document is to describe the interface to the 16AIO API Library and to the underlying Linux device driver. The API Library software provides the interface between "Application Software" and the device driver. The driver software provides the interface between the API Library and the actual 16AIO hardware. The API Library and driver interfaces are based on the board's functionality.

1.2. Acronyms

The following is a list of commonly occurring acronyms which may appear throughout this document.

Acronyms	Description
ADC	Analog-to-Digital Converter
API	Application Programming Interface
BMDMA	Block Mode DMA
DAC	Digital-to-Analog Converter
DMA	Direct Memory Access
GSC	General Standards Corporation
PC104P	This refers to the PC/104+ form factor.
PCI	Peripheral Component Interconnect
PIO	Programmed I/O
PMC	PCI Mezzanine Card

1.3. Definitions

The following is a list of commonly occurring terms which may appear throughout this document.

Term	Definition
...	This is a shortcut representation of the 16AIO installation directory or any of its subdirectories.
16AIO	This is used as a general reference to any board supported by this driver.
API Library	This is a library that provides application-level access to 16AIO hardware.
Application	This is a user mode process, which runs in user space with user mode privileges.
Driver	This is the 16AIO device driver, which is an OS specific executable.
INtime	This refers to the "INtime for Windows" real-time extension for Microsoft Windows. Refer to the <i>16AIO INtime for Windows Driver User Manual</i> .
Library	This is usually a general reference to the API Library.
Linux	This refers to the Linux operating system. Refer to the <i>16AIO Linux Driver User Manual</i> .

1.4. Software Overview

1.4.1. Basic Software Architecture

This section describes the general architecture for the basic components that comprise 16AIO applications. The overall architecture is illustrated in Figure 1 below.

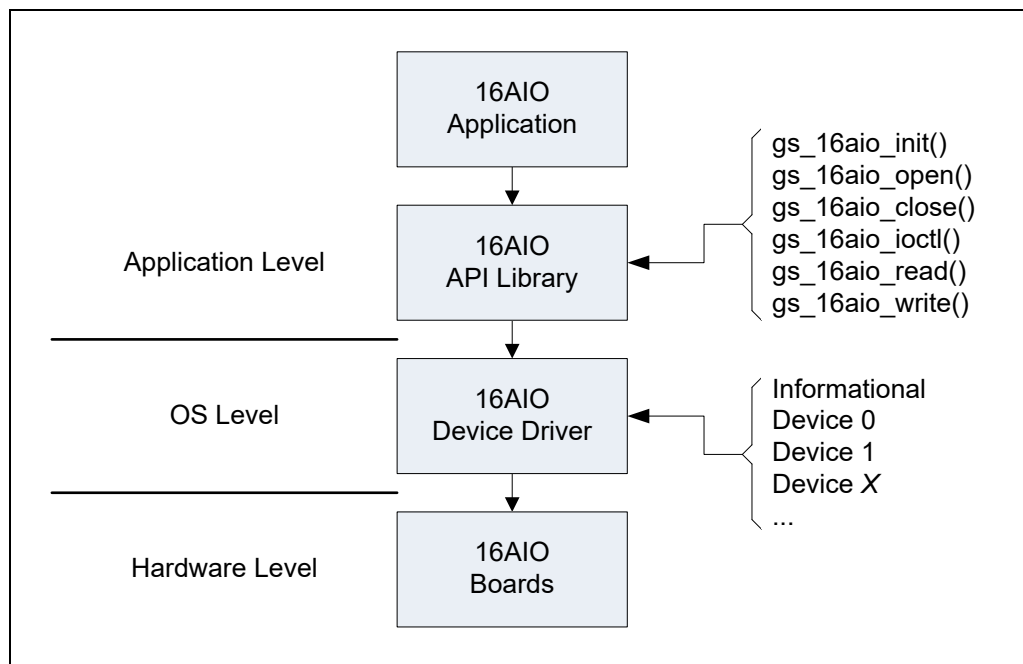


Figure 1 Basic architectural representation.

1.4.2. API Library

The primary means of accessing 16AIO boards is via the 16AIO API Library. This library forms a layer between the application and the driver. Additional information is given in section 4 (page 13). With the library, applications are able to open and close a device and, while open, perform I/O control and read operations.

1.4.3. Device Driver

The device driver is the host software that provides a means of communicating directly with 16AIO hardware. Depending on the OS, the driver may be a user space application, a kernel mode process, or something in between. The software interface to the device driver is analogous to that of the API Library.

1.5. Hardware Overview

The 16AIO is a high-speed analog Input/Output board. The 16AIO offers 16-bits of resolution. The 12AIO offers 12-bits of resolution. The inputs are configurable as either 32 single-ended input channels or as 16 differential input pairs. There are also four analog output channels. The input sampling rate is at an aggregate rate of up to 300,000 samples per second for the 16AIO and it is up to 1,500,000 for the 12AIO. The output sampling rate is up to 300,000 samples per second per channel for the 16AIO and up to 400,000 for the 12AIO. The analog channels can be clocked from either of two independently configurable on-board clocks. Input and output clocking can be either synchronized or independent and can use either on-board or external synchronization signals. A synchronization output is included so that multiple boards can operate in unison. The analog I/O voltage range is software selectable as +/-2.5V, +/-5V or +/-10V. Internal autocalibration networks permit periodic calibration to be performed without removing the board from the system. The board also features two independent 32K deep FIFOs; one for input and one for output. The output FIFO can be configured for single-shot or continuous waveform output. A 16-bit bi-directional digital I/O port is also provided, along with two auxiliary I/O lines. The board also includes DMA and interrupt capabilities.

1.6. Reference Material

The following reference material may be of particular benefit in using the 16AIO, the API Library and the device driver. The specifications provide the information necessary for an in depth understanding of the specialized features implemented on this device.

- The applicable *16AIO Driver User Manual* from General Standards Corporation.
- The applicable *16AIO User Manual* from General Standards Corporation.
- The PCI9080 PCI Bus Master Interface Chip data handbook from PLX Technology, Inc.

PLX Technology Inc.
870 Maude Avenue
Sunnyvale, California 94085 USA
Phone: 1-800-759-3735
WEB: <http://www.plxtech.com>

1.7. Licensing

For licensing information please refer to the text file `LICENSE.txt` in the root installation directory.

1.8. API Naming Conventions, Asynchronous I/O and glibc 2.34

The GSC device driver and API Libraries derive their API identifier prefixes primarily from the base model number of the board(s) supported by the driver. For the 16AIO and 12AIO family of boards, the default API identifiers originally used a prefix of “`aio_`”. Identifier content following the prefix referred more specifically to the identifier’s purpose. So, the identifier “`aio_init`” was a function name whose purpose was to initialize the 16AIO API Library for subsequent use. Unfortunate for the 16AIO API Library, that same prefix has been used by POSIX since 1993 for the Asynchronous I/O services. Not only does the 16AIO API Library use the same prefix, but it duplicates some of the exact same identifiers. Originally, the duplicate identifiers may have generated compilation errors, but only if source code included both the 16AIO API Library header (`16aio.h`) and the Asynchronous I/O header (`aio.h`). Otherwise, the duplicate identifiers may have resulted in either linker errors or run time errors, depending on the environment and other libraries used by the application. Now however, for glibc users, as of glibc 2.34, the Asynchronous I/O functionality has been moved from the Real Time library to the GNU Standard C library. As a result, 16AIO based applications might report multiple definition linker errors or they may operate incorrectly, where such errors had not previously appeared. Thus, such errors might now appear merely by updating the GNU Standard C library. To address these conflicting symbol errors, the 16AIO driver and API Library have both switched from the prefix “`aio_`” to “`gs_16aio_`”, for both lowercase and uppercase identifiers.

2. Installation

For additional information on driver installation refer to this same section number in the OS specific 16AIO driver user manual.

2.1. Host and Environment Support

For information on host and environment support refer to this same section number in the OS specific 16AIO driver user manual.

2.2. Driver and Device Information

Each driver implements an OS specific means of obtaining generic, high-level information about the driver and the installed devices. The information is given in textual format. Each line of text begins with an entry name, which is followed immediately by a colon, a space character, and an entry value. Below is an example of what is provided, followed by descriptions of each entry. This information is accessed by passing a device index value of -1 to the API open service (section 4.6.4, page 16).

```
version: 6.0.117.52
32-bit support: yes
boards: 2
models: 16AIO,12AIO
```

Entry	Description
version	This gives the driver version number in the form x.x.x.x.
32-bit support	This reports the driver's support for 32-bit applications. This will be either "yes" or "no" for 64-bit driver builds and "yes (native)" for 32-bit builds.
boards	This identifies the total number of supported boards the driver detected.
models	This gives a comma separated list of the basic model number for each board the driver detected. The model numbers are listed in the same order that the boards are accessed via the API Library's open function.

The API's source for the text provided is as follows.

OS	Source
Linux	The file "/proc/16aio".
INtime	The Driver Mailbox "16aio".

2.3. File List

For the list of primary files included with each release refer to this same section number in the OS specific 16AIO driver user manual.

2.4. Directory Structure

The following table is representative of the directory structure utilized by each 16AIO driver installation. During the installation process the directory structure is created and populated with the respective files.

NOTE: Additional or alternate directories may be installed, depending on the OS. For additional information refer to this same section number in the OS specific 16AIO driver user manual.

Directory	Description
16aio/	This is the driver root directory. It contains the documentation, the Overall Make Script (section 2.7, page 11) and the below listed subdirectories.
.../api/	This directory contains the API Library source files (section 4, page 13).
.../docsrc/	This directory contains the source files for the code samples given in this document (section 6, page 40).
.../driver/	This directory contains the device driver source files (section 5, page 39).
.../include/	This directory contains the header files for the various libraries.
.../lib/	This directory contains all of the libraries built from the installed sources.
.../samples/	This directory contains the sample application subdirectories and all of their corresponding source files (section 9, page 44).
.../utils/	This directory contains the source files for the utility libraries used by the sample applications (section 7, page 41).

2.5. Installation

For installation instructions refer to this same section number in the OS specific 16AIO driver user manual.

2.6. Removal

For removal instructions refer to this same section number in the OS specific 16AIO driver user manual.

2.7. Overall Make Script

Each 16AIO installation includes an OS specific means of building all of the build targets included in the installation. For additional information refer to this same section number in the OS specific 16AIO driver user manual.

2.8. Environment Variables

For environment variable information refer to this same section number in the OS specific 16AIO driver user manual.

3. Main Interface Files

This section gives general information on the suggested device interface files to use when developing 16AIO based applications.

3.1. Main Header File

Throughout the remainder of this document references are made to various header files included as part of the 16AIO driver installation. For ease of use it is suggested that applications include only the single header file shown below rather than individually including those headers identified separately later in this document. Including this header file pulls in all other pertinent 16AIO specific header files. Therefore, sources may include only this one 16AIO header and make files may reference only this one 16AIO include directory.

Description	File	Location	OS
Header File	16aio_main.h	.../include/	All

3.2. Main Library File

Throughout the remainder of this document references are made to various statically linkable libraries included as part of the 16AIO driver installation. For ease of use it is suggested that applications link only the single library file shown below rather than individually linking those libraries identified separately later in this document. Linking this library file pulls in all other static libraries included with the driver. Therefore, make files may reference only this one 16AIO static library and only this one 16AIO library directory.

Description	File	Location	OS
Library File	16aio_main.a	.../lib/	Linux
	16aio_main.lib	...\\lib\\	INtime

NOTE: For applications using the 16AIO and no other GSC devices, link the 16aio_main.a library. For applications using multiple GSC device types, link the xxxx_main.a library for one of the devices and the xxxx_multi.a library for the others. Linking multiple xxxx_main.a libraries may likely produce link errors due to duplicate symbols being defined. While it may make little or no difference, it is recommended that one choose the xxxx_main.a library from the driver with the largest number in positions three (x.x.X.x.x) and/or four (x.x.x.X.x) in the driver release version number.

NOTE: The 16AIO API Library is implemented as a dynamically loaded library and is thus not linked with the 16AIO Main Library.

3.2.1. Build

For information on building the Main Library refer to this same section number in the OS specific 16AIO driver user manual.

3.2.2. Additional Libraries

For information on any additional required libraries refer to this same section number in the OS specific 16AIO driver user manual.

4. API Library

The 16AIO API Library is the software interface between user applications and the 16AIO device driver. The interface is accessed by including the header file `16aio_api.h`.

NOTE: Contact General Standards Corporation if additional library functionality is required.

4.1. Files

The library files are summarized in the table below.

Description	Files	Location	OS
Source Files	*.c, *.h	.../api/	All
Header File	16aio_api.h	.../include/	All
Library File	lib16aio_api.so †	.../lib/ /usr/lib/	Linux
	16aio_api.lib	...\\lib\\	INtime
	16aio_api.rtl ‡		

† The Linux run time executable is provided in the form of a shared object file.

‡ The INtime run time executable is provided in the form of an INtime DLL.

4.2. Build

For build instructions refer to this same section number in the OS specific 16AIO driver user manual.

4.3. Library Use

For Library usage information refer to this same section number in the OS specific 16AIO driver user manual.

4.4. Macros

The API Library and driver interfaces include the following macros, which are defined in `16aio.h`.

4.4.1. IOCTL Services

The IOCTL macros are documented in section 4.7 (page 19).

4.4.2. Registers

The following gives the complete set of 16AIO registers.

4.4.2.1. GSC Registers

The following table gives the complete set of GSC specific 16AIO registers. Please note that the set of registers supported by any given device may vary according to model and firmware version. For the set of supported registers and their detailed definitions refer to the appropriate *16AIO User Manual*.

NOTE: Refer to the output of the “id” sample application (.../id/) for a complete list of the registers supported by the device being accessed.

Macro	Description
GS_16AIO_GSC_AVR	Autocal Values Register (AVR)
GS_16AIO_GSC_BCR	Board Control Register (BCR)

GS_16AIO_GSC_DIOPR	Digital I/O Port Register (DIOPR)
GS_16AIO_GSC_FRR	Firmware Revision Register (FRR)
GS_16AIO_GSC_IBCR	Input Buffer Control Register (IBCR)
GS_16AIO_GSC_ICR	Interrupt Control Register (ICR)
GS_16AIO_GSC_IDBR	Input Data Buffer Register (IDBR)
GS_16AIO_GSC_OBCR	Output Buffer Control Register (OBCR)
GS_16AIO_GSC_ODBR	Output Data Buffer Register (ODBR)
GS_16AIO_GSC_RGAR	Rate Generator A Register (RGAR)
GS_16AIO_GSC_RGBR	Rate Generator B Register (RGBR)
GS_16AIO_GSC_SSCR	Scan and Sync Control Register (SSCR)

4.4.2.2. PCI Configuration Registers

Access to the PCI registers is seldom required so these registers are not listed here. For the complete list of the PCI register identifiers refer to the driver header file `gsc_pci9080.h`, which is automatically included via `16aio_api.h`.

4.4.2.3. PLX Feature Set Registers

Access to the PLX registers is seldom required so these registers are not listed here. For the complete list of the PLX register identifiers refer to the driver header file `gsc_pci9080.h`, which is automatically included via `16aio_api.h`.

4.5. Data Types

The data types used by the API Library are described with the IOCTL services with which they are used. For additional information refer to section 4.7 (page 19).

4.6. Functions

The interface includes the following functions. The return values reflect the completion status of the requested operation. A return value less than zero always reflects an error condition. The table below summarizes the error status values. For the I/O function, read, non-negative return values reflect the number of bytes transferred between the application and the interface. A value equal to the requested transfer size indicates complete success. Return values less than the requested transfer size indicate that the I/O timeout expired. For the other API function calls a return value of zero indicates success.

Return Value	Description	OS
-1 to -999	This is the value “(-errno)” (see <code>errno.h</code>).	All
<= -1000	This is the value “(-(int) (GetLastRtError()+1000))”.	INtime

4.6.1. `gs_16aio_close()`

This function is the entry point to close a connection made via the API's open call. The device is put in an initialized state before this call returns. This function is multithread safe.

Prototype

```
int gs_16aio_close(int fd);
```

Argument	Description
fd	This is the file descriptor obtained from the open service (section 4.6.4, page 16).

Return Value	Description
0	The operation succeeded.
< 0	An error occurred. See error value description above.

Example

```
#include <stdio.h>

#include "16aio_dsl.h"

int gs_16aio_close_dsl(int fd)
{
    int errs;
    int ret;

    ret = gs_16aio_close(fd);

    if (ret)
        printf("ERROR: gs_16aio_close() returned %d\n", ret);

    errs    = ret ? 1 : 0;
    return(errs);
}
```

4.6.2. gs_16aio_init()

This function is the entry point to initializing the 16AIO API Library and must be the first call into the Library. This function may be called more than once, but only the first successful call actually initializes the library. Subsequent calls perform no operation at all. All other API calls return a failure status when the API Library is not initialized.

Prototype

```
int gs_16aio_init(void);
```

Return Value	Description
0	The operation succeeded.
< 0	An error occurred. See error value description above.

Example

```
#include <stdio.h>

#include "16aio_dsl.h"

int gs_16aio_init_dsl(void)
{
    int errs;
    int ret;

    ret = gs_16aio_init();

    if (ret)
        printf("ERROR: gs_16aio_init() returned %d\n", ret);

    errs    = ret ? 1 : 0;
```

```

    return(errs);
}

```

4.6.3. gs_16aio_ioctl()

This function is the entry point to performing setup and control operations on a 16AIO. This function should only be called after a successful open of the respective device. The specific operation performed varies according to the `request` argument. The `request` argument also governs the use and interpretation of the `arg` argument. The set of supported options for the `request` argument consists of the IOCTL services supported by the driver, which are defined in section 4.7 (page 19). This function is multithread safe.

NOTE: IOCTL operations are not supported for an open on device index `-1`.

Prototype

```
int gs_16aio_ioctl(int fd, int request, void* arg);
```

Argument	Description
<code>fd</code>	This is the file descriptor obtained from the open service (section 4.6.4, page 16).
<code>request</code>	This specifies the desired operation to be performed (section 4.7, page 19).
<code>arg</code>	This is specific to the IOCTL operation specified by the <code>request</code> argument.

Return Value	Description
<code>0</code>	The operation succeeded.
<code>< 0</code>	An error occurred. See error value description above.

Example

```

#include <stdio.h>

#include "16aio_dsl.h"

int gs_16aio_ioctl_dsl(int fd, int request, void* arg)
{
    int errs;
    int ret;

    ret = gs_16aio_ioctl(fd, request, arg);

    if (ret)
        printf("ERROR: gs_16aio_ioctl() returned %d\n", ret);

    errs    = ret ? 1 : 0;
    return(errs);
}

```

4.6.4. gs_16aio_open()

This function is the entry point to open a connection to a 16AIO board. Before returning, the initialize IOCTL service is called to reset all hardware and software settings to their defaults. This function is multithread safe.

Prototype

```
int gs_16aio_open(int device, int share, int* fd);
```


Argument	Description						
device	This is the zero-based index of the 16AIO to access. †						
share	Open the device in Shared Access Mode? If non-zero the device is opened in Shared Access Mode (see below). If zero the device is opened in Exclusive Access Mode (see below).						
fd	The device handle is returned here. The pointer cannot be NULL. Values returned are as follows. <table border="1"> <tr> <th>Value</th><th>Description</th></tr> <tr> <td>>= 0</td><td>This is the handle to use to access the device in subsequent calls.</td></tr> <tr> <td>-1</td><td>There was an error. The device is not accessible.</td></tr> </table>	Value	Description	>= 0	This is the handle to use to access the device in subsequent calls.	-1	There was an error. The device is not accessible.
Value	Description						
>= 0	This is the handle to use to access the device in subsequent calls.						
-1	There was an error. The device is not accessible.						

† The index value -1 can also be given to acquire driver information (section 2.2, page 10).

Return Value	Description
0	The operation succeeded.
< 0	An error occurred. See error value description above.

Example

```
#include <stdio.h>

#include "16aio_dsl.h"

int gs_16aio_open_dsl(int device, int share, int* fd)
{
    int errs;
    int ret;

    ret = gs_16aio_open(device, share, fd);

    if (ret)
        printf("ERROR: gs_16aio_open() returned %d\n", ret);

    errs    = ret ? 1 : 0;
    return(errs);
}
```

4.6.4.1. Access Modes

The value of the `share` argument determines the device access mode, as follows.

Shared Access Mode:

Shared Access Mode allows multiple applications to access the same device simultaneously. In this mode, the first successful open request returns with the device in an initialized state. Subsequent successful Shared Access Mode open requests do not affect the state of the device. Once opened in Shared Access Mode, the device access remains in this mode until all Shared Access Mode accesses release the device with a close request.

Exclusive Access Mode:

Exclusive Access Mode allows a single application to acquire exclusive access to a device. In this mode, a successful open request returns with the device in an initialized state. While open in this mode all subsequent open requests will fail regardless of the access mode requested. Once opened in Exclusive Access Mode, the device access remains in this mode until released by the application with a close request.

4.6.5. `gs_16aio_read()`

This function is the entry point to reading data from an open connection. This function should only be called after a successful open of the respective device. The function reads up to `bytes` bytes. This function is multithread safe.

NOTE: If an open was performed using an index of `-1`, then read requests will acquire information from the driver (section 2.2, page 10) rather than data from a device.

NOTE: For additional information refer to the Data Transfer Modes section (section 8.4, page 43).

Prototype

```
int gs_16aio_read(int fd, void* dst, size_t bytes);
```

Argument	Description
<code>fd</code>	This is the file descriptor obtained from the open service (section 4.6.4, page 16).
<code>dst</code>	The data read is put here.
<code>bytes</code>	This is the desired number of bytes to read. When reading from a device, this must be a multiple of four (4).

Return Value	Description
0 to <code>bytes</code>	The operation succeeded. When reading from a device, a value less than <code>bytes</code> indicates that the I/O timeout period lapsed (section 4.7.43, page 34) before the entire request could be satisfied.
< 0	An error occurred. See error value description above.

Example

```
#include <stdio.h>

#include "16aio_dsl.h"

int gs_16aio_read_dsl(int fd, void* dst, size_t bytes, size_t* qty)
{
    int errs;
    int ret;

    ret = gs_16aio_read(fd, dst, bytes);

    if (ret < 0)
        printf("ERROR: gs_16aio_read() returned %d\n", ret);

    if (qty)
        qty[0] = (ret < 0) ? 0 : (size_t) ret;

    errs = (ret < 0) ? 1 : 0;
    return(errs);
}
```

4.6.6. `gs_16aio_write()`

This function is the entry point to writing data to an open connection. This function should only be called after a successful open of the respective device. The function writes up to `bytes` bytes to the board. This function is multithread safe.

NOTE: The driver's write service may dynamically manipulate the output buffer threshold level. When this is done the original value will be restored before the write service returns. The output buffer threshold level will not be manipulated if the output buffer threshold status has been selected as an interrupt source. In these cases, write performance may be reduced.

Prototype

```
int gs_16aio_write(int fd, const void* src, size_t bytes);
```

Argument	Description
fd	This is the file descriptor obtained from the open service (section 4.6.4, page 16).
src	The data to write is taken from this pointer.
bytes	This is the desired number of bytes to write. This must be a multiple of four (4).

Return Value	Description
0 to bytes	The operation succeeded. When writing to a device, a value less than bytes indicates that the I/O timeout period lapsed (section 4.7.46, page 35) before the entire request could be satisfied.
< 0	An error occurred. See error value description above.

Example

```
#include <stdio.h>

#include "16aio_dsl.h"

int gs_16aio_write_dsl(int fd, const void* src, size_t bytes, size_t* qty)
{
    int errs;
    int ret;

    ret = gs_16aio_write(fd, src, bytes);

    if (ret < 0)
        printf("ERROR: gs_16aio_write() returned %d\n", ret);

    if (qty)
        qty[0] = (ret < 0) ? 0 : (size_t) ret;

    errs = (ret < 0) ? 1 : 0;
    return(errs);
}
```

4.7. IOCTL Services

The 16AIO API Library and device driver implement the following IOCTL services. Each service is described along with the applicable `gs_16aio_ioctl()` function arguments.

4.7.1. GS_16AIO_IOCTL_AI_BUF_CLEAR

This service immediately clears the current content from the input buffer. This service does not halt input sampling.

Usage

Argument	Description
request	GS_16AIO_IOCTL_AI_BUF_CLEAR
arg	Not used.

4.7.2. GS_16AIO_IOCTL_AI_BUF_THR_LVL

This service configures the input buffer threshold level.

Usage

Argument	Description
request	GS_16AIO_IOCTL_AI_BUF_THR_LVL
arg	s32*

Valid argument values are from zero to 0x7FFF, and -1. A value of -1 will return the current threshold level setting.

4.7.3. GS_16AIO_IOCTL_AI_BUF_THR_STS

This service retrieves the current input buffer threshold level status, which indicates whether or not there is more than Input Buffer Threshold Level number of 32-bit data items in the input buffer.

Usage

Argument	Description
request	GS_16AIO_IOCTL_AI_BUF_THR_STS
arg	s32*

The current status is reported as one of the following values.

Value	Description
GS_16AIO_AI_BUF_THR_STS_CLEAR	The input buffer contains Threshold Level number of data items, or fewer.
GS_16AIO_AI_BUF_THR_STS_SET	The input buffer contains more than Threshold Level number of data items.

4.7.4. GS_16AIO_IOCTL_AI_MODE

This service retrieves the current input buffer mode, which specifies the analog input data source.

Usage

Argument	Description
request	GS_16AIO_IOCTL_AI_MODE
arg	s32*

Valid argument values are as follows.

Value	Description
-1	Retrieve the current setting.
GS_16AIO_AI_MODE_AO_0	This refers to analog output channel 0.
GS_16AIO_AI_MODE_AO_1	This refers to analog output channel 1.

GS_16AIO_AI_MODE_AO_2	This refers to analog output channel 2.
GS_16AIO_AI_MODE_AO_3	This refers to analog output channel 3.
GS_16AIO_AI_MODE_DIFF	This refers to the differential inputs, which limits the number of input channels to 16.
GS_16AIO_AI_MODE_SINGLE	This refers to the single ended inputs, which expands the number of input channels to 32.
GS_16AIO_AI_MODE_VREF	This refers to the VREF voltage input source.
GS_16AIO_AI_MODE_ZERO	This refers to the zero-voltage input source.

4.7.5. GS_16AIO_IOCTL_AI_SCAN_1_CHAN

This service selects the channel to use when the Analog Input Mode is set to the Single Channel option.

Usage

Argument	Description
request	GS_16AIO_IOCTL_AI_SCAN_1_CHAN
arg	s32*

Valid argument values are from zero to one less than the number of input channels, and -1. The number of input channels is 32 for single ended mode and 16 for differential mode. A value of -1 will return the current threshold level setting.

4.7.6. GS_16AIO_IOCTL_AI_SCAN_CLK_SRC

This service configures the source for the analog input sampling clock.

Usage

Argument	Description
request	GS_16AIO_IOCTL_AI_SCAN_CLK_SRC
arg	s32*

Valid argument values are as follows.

Value	Description
-1	Retrieve the current setting.
GS_16AIO_AI_SCAN_CLK_SRC_BCR	This refers to the Input Sync bit in the BCR.
GS_16AIO_AI_SCAN_CLK_SRC_EXT	This refers to the external clock source.
GS_16AIO_AI_SCAN_CLK_SRC_RGA	This refers to the Rate Generator A.
GS_16AIO_AI_SCAN_CLK_SRC_RGB	This refers to the Rate Generator B.

4.7.7. GS_16AIO_IOCTL_AI_SCAN_SIZE

This service configures the selection for the number of input channels included in a scan.

Usage

Argument	Description
request	GS_16AIO_IOCTL_AI_SCAN_SIZE
arg	s32*

Valid argument values are as follows.

Value	Description
-1	Retrieve the current setting.
GS_16AIO_AI_SCAN_SIZE_0_1	This refers to scanning input channels zero through one.
GS_16AIO_AI_SCAN_SIZE_0_3	This refers to scanning input channels zero through three.
GS_16AIO_AI_SCAN_SIZE_0_7	This refers to scanning input channels zero through seven.
GS_16AIO_AI_SCAN_SIZE_0_15	This refers to scanning input channels zero through 15.
GS_16AIO_AI_SCAN_SIZE_0_31	This refers to scanning input channels zero through 31. This option should not be made when using differential mode operation.
GS_16AIO_AI_SCAN_SIZE_SINGLE	This refers to scanning a single input channel.

4.7.8. GS_16AIO_IOCTL_AI_SYNC

This service initiates an input sync operation. The driver returns immediately rather than waiting for the operation to complete.

Usage

Argument	Description
request	GS_16AIO_IOCTL_AI_SYNC
arg	Not used.

4.7.9. GS_16AIO_IOCTL_AO_BUF_CLEAR

This service immediately clears the current content from the output buffer.

Usage

Argument	Description
request	GS_16AIO_IOCTL_AO_BUF_CLEAR
arg	Not used.

4.7.10. GS_16AIO_IOCTL_AO_BUF_THR_LVL

This service configures the output buffer threshold level.

Usage

Argument	Description
request	GS_16AIO_IOCTL_AO_BUF_THR_LVL
arg	s32*

Valid argument values are from zero to 0x7FFF, and -1. A value of -1 will return the current threshold level setting.

4.7.11. GS_16AIO_IOCTL_AO_BUF_THR_STS

This service retrieves the current output buffer threshold level status, which indicates whether or not there is more than output Buffer Threshold Level number of 32-bit data items in the input buffer.

Usage

Argument	Description
request	GS_16AIO_IOCTL_AO_BUF_THR_STS
arg	s32*

The current status is reported as one of the following values.

Value	Description
GS_16AIO_AO_BUF_THR_STS_CLEAR	The output buffer contains Threshold Level number of data items, or fewer.
GS_16AIO_AO_BUF_THR_STS_SET	The output buffer contains more than Threshold Level number of data items.

4.7.12. GS_16AIO_IOCTL_AO_BURST_CLK_SRC

This service configures the source for the analog output bursting clock.

Usage

Argument	Description
request	GS_16AIO_IOCTL_AO_BURST_CLK_SRC
arg	s32*

Valid argument values are as follows.

Value	Description
-1	Retrieve the current setting.
GS_16AIO_AO_BURST_CLK_SRC_BCR	This refers to the Output Sync bit in the BCR.
GS_16AIO_AO_BURST_CLK_SRC_EXT	This refers to the external clock source.
GS_16AIO_AO_BURST_CLK_SRC_RGA	This refers to the Rate Generator A.
GS_16AIO_AO_BURST_CLK_SRC_RGB	This refers to the Rate Generator B.

4.7.13. GS_16AIO_IOCTL_AO_BURST_ENABLE

This service enables or disables output bursting.

Usage

Argument	Description
request	GS_16AIO_IOCTL_AO_BURST_ENABLE
arg	s32*

Valid argument values are as follows.

Value	Description
-1	Retrieve the current setting.
GS_16AIO_AO_BURST_ENABLE_NO	This refers to bursting being disabled.
GS_16AIO_AO_BURST_ENABLE_YES	This refers to bursting being enabled.

4.7.14. GS_16AIO_IOCTL_AO_CLK_SRC

This service configures the source for the analog output sampling clock.

Usage

Argument	Description
request	GS_16AIO_IOCTL_AO_CLK_SRC
arg	s32*

Valid argument values are as follows.

Value	Description
-1	Retrieve the current setting.
GS_16AIO_AO_CLK_SRC_DISABLE	This disables analog output.
GS_16AIO_AO_CLK_SRC_EXT	This refers to the external clock source.
GS_16AIO_AO_CLK_SRC_RGA	This refers to the Rate Generator A.
GS_16AIO_AO_CLK_SRC_RGB	This refers to the Rate Generator B.

4.7.15. GS_16AIO_IOCTL_AO_LOOPING

This service enables or disabled analog output recycling for repetitive pattern generation.

Usage

Argument	Description
request	GS_16AIO_IOCTL_AO_LOOPING
arg	s32*

Valid argument values are as follows.

Value	Description
-1	Retrieve the current setting.
GS_16AIO_AO_LOOPING_DISABLE	This disables output data recycling.
GS_16AIO_AO_LOOPING_ENABLE	This enables output data recycling.

4.7.16. GS_16AIO_IOCTL_AO_SYNC

This service initiates an output sync operation. The driver returns immediately rather than waiting for the operation to complete.

Usage

Argument	Description
request	GS_16AIO_IOCTL_AO_SYNC
arg	Not used.

4.7.17. GS_16AIO_IOCTL_AO_TIMING

This service configures the relative timing at which analog output is posted to the cable interface.

Usage

Argument	Description
request	GS_16AIO_IOCTL_AO_TIMING
arg	s32*

Valid argument values are as follows.

Value	Description
-1	Retrieve the current setting.
GS_16AIO_AO_TIMING_SEQ	This refers to sequential operation.
GS_16AIO_AO_TIMING_SIMUL	This refers to simultaneous operation.

4.7.18. GS_16AIO_IOCTL_AUTOCAL

This service initiates an autocalibration cycle. The driver waits for the operation to complete before returning.

NOTE: This service overwrites the current interrupt selection in order to detect the Autocalibration Done interrupt.

NOTE: When an error is encountered, the service writes a brief, descriptive error message to the system log.

Usage

Argument	Description
request	GS_16AIO_IOCTL_AUTOCAL
arg	Not used.

4.7.19. GS_16AIO_IOCTL_AUTOCAL_STATUS

This service retrieves the results of the most recent autocalibration cycle.

Usage

Argument	Description
request	GS_16AIO_IOCTL_AUTOCAL_STATUS
arg	s32*

The current status is reported as one of the following values.

Value	Description
GS_16AIO_AUTOCAL_STATUS_ACTIVE	Autocalibration is still in progress.
GS_16AIO_AUTOCAL_STATUS_FAIL	Autocalibration failed.
GS_16AIO_AUTOCAL_STATUS_PASS	Autocalibration passed.

4.7.20. GS_16AIO_IOCTL_AUX_READ

This service returns the current input level at the cable's Auxiliary Input.

Usage

Argument	Description
request	GS_16AIO_IOCTL_AUX_READ
arg	s32*

Argument values returned are zero or one.

4.7.21. GS_16AIO_IOCTL_AUX_WRITE

This service applies a value to the cable's Auxiliary Output.

Usage

Argument	Description
request	GS_16AIO_IOCTL_AUX_WRITE
arg	s32*

Valid argument values are from zero or one, and -1. A value of -1 will return the current output state.

4.7.22. GS_16AIO_IOCTL_DATA_FORMAT

This service configures and retrieves the current data encoding format.

Usage

Argument	Description
request	GS_16AIO_IOCTL_DATA_FORMAT
arg	s32*

Valid argument values are as follows.

Value	Description
-1	Retrieve the current setting.
GS_16AIO_DATA_FORMAT_2S_COMP	This refers to twos compliment encoding.
GS_16AIO_DATA_FORMAT_OFF_BIN	This refers to offset binary encoding.

4.7.23. GS_16AIO_IOCTL_DIO_DIR_OUT

This service configures the direction of the two byte-wide digital I/O ports. If a bit is set then the corresponding byte is configured as an output. If a bit is clear then the byte is configured as an input. Both ports default to inputs. The auxiliary input and output port bits are unaffected.

Usage

Argument	Description
request	GS_16AIO_IOCTL_DIO_DIR_OUT
arg	s32*

Valid argument values are from zero to 0x3, and -1. A value of -1 will return the current output state. The 0x1 bit refers to the lower byte and the 0x2 bit refers to the upper byte.

4.7.24. GS_16AIO_IOCTL_DIO_READ

This service returns the current digital port value applied at the cable interface. This does not include the auxiliary input or output ports. Input port bits reflect the value being applied externally to the cable interface. Output port bits reflect the value being applied by the board to the cable interface.

Usage

Argument	Description
request	GS_16AIO_IOCTL_DIO_READ
arg	s32*

Argument values returned are from zero to 0xFFFF.

4.7.25. GS_16AIO_IOCTL_DIO_WRITE

This service updates the digital output port value applied by the board to the cable interface for the byte wide ports operating as outputs. This does not include the auxiliary input or output ports.

Usage

Argument	Description
request	GS_16AIO_IOCTL_DIO_WRITE
arg	s32*

Valid argument values are from zero to 0x2FFFF, and -1. A value of -1 will return the current output port value, which may not reflect current overall port value for those bits operating as inputs.

4.7.26. GS_16AIO_IOCTL_EXT_SYNC_SRC

This service configures the source for the external sync output signal.

Usage

Argument	Description
request	GS_16AIO_IOCTL_EXT_SYNC_SRC
arg	s32*

Valid argument values are as follows.

Value	Description
-1	Retrieve the current setting.
GS_16AIO_AI_EXT_SYNC_SRC_AISC	This refers to the analog input scan clock.
GS_16AIO_AI_EXT_SYNC_SRC_AOS	This refers to the analog output sync clock.
GS_16AIO_AI_EXT_SYNC_SRC_EXT	This refers to the external sync input signal.
GS_16AIO_AI_EXT_SYNC_SRC_DISABLE	This disables the external sync output.

4.7.27. GS_16AIO_IOCTL_INITIALIZE

This service resets all hardware and software settings to their defaults.

NOTE: If the initialization service returns an error status, an error message will be posted to the system log briefly describing the error condition.

Usage

Argument	Description
request	GS_16AIO_IOCTL_INITIALIZE
arg	Not used.

4.7.28. GS_16AIO_IOCTL_IRQ0_SEL

This service configures the interrupting source for interrupt option zero.

Usage

Argument	Description
request	GS_16AIO_IOCTL_IRQ0_SEL
arg	s32*

Valid argument values are as follows.

Value	Description
-1	Retrieve the current setting.
GS_16AIO_IRQ0_AUTOCAL_DONE	This refers to the completion of an autocalibration cycle.
GS_16AIO_IRQ0_AUX_IN_H2L	This refers to a high-to-low transition on the Auxiliary Input.
GS_16AIO_IRQ0_AUX_IN_L2H	This refers to a low-to-high transition on the Auxiliary Input.
GS_16AIO_IRQ0_IDLE_INIT	This refers to the completion of an initialization cycle.

4.7.29. GS_16AIO_IOCTL_IRQ1_SEL

This service configures the interrupting source for interrupt option one.

Usage

Argument	Description
request	GS_16AIO_IOCTL_IRQ1_SEL
arg	s32*

Valid argument values are as follows.

Value	Description
-1	Retrieve the current setting.
GS_16AIO_IRQ1_AI_BUF_THR_H2L	This refers to a high-to-low transition on the Analog Input Buffer Threshold Status.
GS_16AIO_IRQ1_AI_BUF_THR_L2H	This refers to a low-to-high transition on the Analog Input Buffer Threshold Status.
GS_16AIO_IRQ1_IDLE	This option disables the interrupt.

4.7.30. GS_16AIO_IOCTL_IRQ2_SEL

This service configures the interrupting source for interrupt option two.

Usage

Argument	Description
request	GS_16AIO_IOCTL_IRQ2_SEL
arg	s32*

Valid argument values are as follows.

Value	Description
-1	Retrieve the current setting.
GS_16AIO_IRQ2_AO_BUF_THR_H2L	This refers to a high-to-low transition on the Analog Output Buffer Threshold Status.
GS_16AIO_IRQ2_AO_BUF_THR_L2H	This refers to a low-to-high transition on the Analog Output Buffer Threshold Status.

GS_16AIO_IRQ2_AO_BURST_DONE	This refers to the completion of Analog Output burst operation.
GS_16AIO_IRQ2_IDLE	This option disables the interrupt.

4.7.31. GS_16AIO_IOCTL_QUERY

This service is used to query the driver for various pieces of information about the driver and the board. The item being queried is supplied as the argument value. The argument value is updated with the response.

Usage

Argument	Description
request	GS_16AIO_IOCTL_QUERY
arg	s32*

Valid argument values are as follows.

Value	Description
GS_16AIO_QUERY_AUTOCAL_MS	This is the duration of an autocalibration cycle in milliseconds.
GS_16AIO_QUERY_COUNT	This is the number of different query options recognized by the driver.
GS_16AIO_QUERY_DEVICE_TYPE	This is the device type and should equal GSC_DEV_TYPE_16AIO or GSC_DEV_TYPE_12AIO.
GS_16AIO_QUERY_FGEN_AI_MAX	This is the maximum Analog Input rate generator output in hertz.
GS_16AIO_QUERY_FGEN_AI_MIN	This is the minimum Analog Input rate generator output in hertz.
GS_16AIO_QUERY_FGEN_AO_MAX	This is the maximum Analog Output rate generator output in hertz.
GS_16AIO_QUERY_FGEN_AO_MIN	This is the minimum Analog Output rate generator output in hertz.
GS_16AIO_QUERY_FIFO_SIZE_RX	This is the capacity of the input FIFO in 32-bit samples.
GS_16AIO_QUERY_FIFO_SIZE_TX	This is the capacity of the output FIFO in 32-bit samples.
GS_16AIO_QUERY_FREF_DEFAULT	This is the default master clock frequency in hertz.
GS_16AIO_QUERY_FSAMP_AI_MAX	This is the maximum Analog Input sample rate in samples per second.
GS_16AIO_QUERY_FSAMP_AI_MIN	This is the minimum Analog Input sample rate in samples per second.
GS_16AIO_QUERY_FSAMP_AO_MAX	This is the maximum Analog Output sample rate in samples per second.
GS_16AIO_QUERY_FSAMP_AO_MIN	This is the minimum Analog Output sample rate in samples per second.
GS_16AIO_QUERY_INIT_MS	This is the duration of an initialization cycle in milliseconds.
GS_16AIO_QUERY_NRATE_AI_MASK	This is the mask of valid Analog Input NRATE rate generator divisor bits.
GS_16AIO_QUERY_NRATE_AI_MAX	This is the maximum Analog Input NRATE rate generator divisor.
GS_16AIO_QUERY_NRATE_AI_MIN	This is the minimum Analog Input NRATE rate generator divisor.
GS_16AIO_QUERY_NRATE_AO_MASK	This is the mask of valid Analog Output NRATE rate generator divisor bits.
GS_16AIO_QUERY_NRATE_AO_MAX	This is the maximum Analog Output NRATE rate generator divisor.

GS_16AIO_QUERY_NRATE_AO_MIN	This is the minimum Analog Output NRATE rate generator divisor.
GS_16AIO_QUERY_RES_BITS	This is the number of analog conversion resolution bits.

Valid return values are as indicated in the above table and as given in the below table.

Value	Description
GS_16AIO_IOCTL_QUERY_ERROR	Either there was a processing error or the query option is unrecognized.

4.7.32. GS_16AIO_IOCTL_RANGE

This service configures the analog input and output voltage ranges.

Usage

Argument	Description
request	GS_16AIO_IOCTL_RANGE
arg	s32*

Valid argument values are as follows.

Value	Description
-1	Retrieve the current setting.
GS_16AIO_RANGE_2_5V	This refers to the range of ± 2.5 volts.
GS_16AIO_RANGE_5V	This refers to the range of ± 5 volts.
GS_16AIO_RANGE_10V	This refers to the range of ± 10 volts.

Those 32 channel boards ordered with the -16HV60V option support an alternate voltage range on the upper 16 input channels. Software is not able to detect when this option is present. In addition, the alternate voltage range is not selectable independent of the voltage range for the lower 16 input channels. That is, the voltage range for the upper 16 input channels is based on the voltage range selected for the lower 16 input channels. For boards with this option, the argument values and resulting voltage range selections are as follows. The output channel voltage ranges are unaffected and their voltage ranges are per the above table.

Value	Input Voltage Ranges	
	Lower 16 Channels	Upper 16 Channels
GS_16AIO_RANGE_2_5V	± 2.5 volts	± 15 volts
GS_16AIO_RANGE_5V	± 5 volts	± 30 volts
GS_16AIO_RANGE_10V	± 10 volts	± 60 volts

4.7.33. GS_16AIO_IOCTL_REG_MOD

This service performs a read-modify-write of a 16AIO register. This includes only the GSC firmware registers. The PCI and PLX Feature Set Registers are read-only. Refer to `16aio.h` for the complete list of GSC firmware registers.

Usage

Argument	Description
request	GS_16AIO_IOCTL_REG_MOD
arg	gsc_reg_t*

Definition

```
typedef struct
{
    u32 reg;
    u32 value;
    u32 mask;
} gsc_reg_t;
```

Fields	Description
reg	This is set to the identifier for the register to access.
value	This contains the value for the register bits to modify.
mask	This specifies the set of bits to modify. If a bit here is set, then the respective register bits is modified. If a bit here is zero, then the respective register bit is unmodified.

4.7.34. GS_16AIO_IOCTL_REG_READ

This service reads the value of a 16AIO register. This includes the PCI registers, the PLX Feature Set Registers and the GSC firmware registers. Refer to `16aio.h` and `gsc_pci9080.h` for the complete list of accessible registers.

Usage

Argument	Description
request	GS_16AIO_IOCTL_REG_READ
arg	<code>gsc_reg_t*</code>

Definition

```
typedef struct
{
    u32 reg;
    u32 value;
    u32 mask;
} gsc_reg_t;
```

Fields	Description
reg	This is set to the identifier for the register to access.
value	This is the value read from the specified register.
mask	This is ignored for read request.

4.7.35. GS_16AIO_IOCTL_REG_WRITE

This service writes a value to a 16AIO register. This includes only the GSC firmware registers. The PCI and PLX Feature Set Registers are read-only. Refer to `16aio.h` for a complete list of the GSC firmware registers.

Usage

Argument	Description
request	GS_16AIO_IOCTL_REG_WRITE
arg	<code>gsc_reg_t*</code>

Definition

```
typedef struct
```

```

{
    u32 reg;
    u32 value;
    u32 mask;
} gsc_reg_t;

```

Fields	Description
reg	This is set to the identifier for the register to access.
value	This is the value to write to the specified register.
mask	This is ignored for write request.

4.7.36. GS_16AIO_IOCTL_RGA_ENABLE

This service enables or disables the Rate Generator A.

Usage

Argument	Description
request	GS_16AIO_IOCTL_RGA_ENABLE
arg	s32*

Valid argument values are as follows.

Value	Description
-1	Retrieve the current setting.
GS_16AIO_GEN_ENABLE_NO	This disables the rate generator.
GS_16AIO_GEN_ENABLE_YES	This enables the rate generator.

4.7.37. GS_16AIO_IOCTL_RGA_NRATE

This service configures Rate Generator A NRATE divider value.

Usage

Argument	Description
request	GS_16AIO_IOCTL_RGA_NRATE
arg	s32*

Valid argument values are from two to 0xFFFF, and -1. For non-cascaded operation, the minimum valid value is 80. A value of -1 will return the current divider setting.

4.7.38. GS_16AIO_IOCTL_RGB_CLK_SRC

This service configures the clock source for the Rate Generator B.

Usage

Argument	Description
request	GS_16AIO_IOCTL_RGB_CLK_SRC
arg	s32*

Valid argument values are as follows.

Value	Description
-1	Retrieve the current setting.
GS_16AIO_RGB_CLK_SRC_MASTER	This refers to the master clock.
GS_16AIO_RGB_CLK_SRC_RGA	This refers to the Rate Generator A.

4.7.39. GS_16AIO_IOCTL_RGB_ENABLE

This service enables or disables the Rate Generator B.

Usage

Argument	Description
request	GS_16AIO_IOCTL_RGB_ENABLE
arg	s32*

Valid argument values are as follows.

Value	Description
-1	Retrieve the current setting.
GS_16AIO_GEN_ENABLE_NO	This disables the rate generator.
GS_16AIO_GEN_ENABLE_YES	This enables the rate generator.

4.7.40. GS_16AIO_IOCTL_RGB_NRATE

This service configures Rate Generator B NRATE divider value.

Usage

Argument	Description
request	GS_16AIO_IOCTL_RGB_NRATE
arg	s32*

Valid argument values are from two to 0xFFFF, and -1. For non-cascaded operation, the minimum valid value is 80. A value of -1 will return the current divider setting.

4.7.41. GS_16AIO_IOCTL_RX_IO_ABORT

This service aborts an ongoing read request.

Usage

Argument	Description
request	GS_16AIO_IOCTL_RX_IO_ABORT
arg	s32*

The results are reported as one of the following values.

Value	Description
GS_16AIO_IO_ABORT_NO	A read request was not aborted as none were ongoing.
GS_16AIO_IO_ABORT_YES	A read request was aborted.

4.7.42. GS_16AIO_IOCTL_RX_IO_MODE

This service sets the I/O mode used for data read requests.

Usage

Argument	Description
request	GS_16AIO_IOCTL_RX_IO_MODE
arg	s32*

Valid argument values are as follows.

Value	Description
-1	Retrieve the current setting.
GSC_IO_MODE_BMDMA	Use Block Mode DMA.
GSC_IO_MODE_PIO	Use PIO mode, which is repetitive register access. This is the default.

4.7.43. GS_16AIO_IOCTL_RX_IO_TIMEOUT

This service sets the timeout limit for read requests. The value is expressed in seconds.

Usage

Argument	Description
request	GS_16AIO_IOCTL_RX_IO_TIMEOUT
arg	s32*

Valid argument values are in the range from zero to 3600, -1, and GS_16AIO_IOCTL_TIMEOUT_INFINITE. A value of zero tells the driver not to sleep in order to wait for more data, and should only be used with PIO mode reads. A value of -1 is used to retrieve the current setting. If the option GS_16AIO_IOCTL_TIMEOUT_INFINITE is used, then the driver will wait indefinitely rather than timing out. The default is 10 seconds.

4.7.44. GS_16AIO_IOCTL_TX_IO_ABORT

This service aborts an ongoing write request.

Usage

Argument	Description
request	GS_16AIO_IOCTL_TX_IO_ABORT
arg	s32*

The results are reported as one of the following values.

Value	Description
GS_16AIO_IOCTL_ABORT_NO	A write request was not aborted as none were ongoing.
GS_16AIO_IOCTL_ABORT_YES	A write request was aborted.

4.7.45. GS_16AIO_IOCTL_TX_IO_MODE

This service sets the I/O mode used for data write requests.

Usage

Argument	Description
request	GS_16AIO_IOCTL_TX_IO_MODE
arg	s32*

Valid argument values are as follows.

Value	Description
-1	Retrieve the current setting.
GSC_IO_MODE_BMDMA	Use Block Mode DMA.
GSC_IO_MODE_PIO	Use PIO mode, which is repetitive register access. This is the default.

4.7.46. GS_16AIO_IOCTL_TX_IO_TIMEOUT

This service sets the timeout limit for write requests. The value is expressed in seconds.

Usage

Argument	Description
request	GS_16AIO_IOCTL_TX_IO_TIMEOUT
arg	s32*

Valid argument values are in the range from zero to 3600, -1, and GS_16AIO_IOCTL_TIMEOUT_INFINITE. A value of zero tells the driver not to sleep in order to wait for more space, and should only be used with PIO mode reads. A value of -1 is used to retrieve the current setting. If the option GS_16AIO_IOCTL_TIMEOUT_INFINITE is used, then the driver will wait indefinitely rather than timing out. The default is 10 seconds.

4.7.47. GS_16AIO_IOCTL_WAIT_CANCEL

This service resumes all threads blocked via GS_16AIO_IOCTL_WAIT_EVENT IOCTL calls (section 4.7.48, page 36), according to the provided criteria. When a blocked thread is waiting for any event specified in the structure, then the thread is resumed.

NOTE: The driver itself makes use of the wait services for various internal operations. Driver initiated waits are unaffected by application cancel requests.

Usage

Argument	Description
request	GS_16AIO_IOCTL_WAIT_CANCEL
arg	gsc_wait_t*

Definition

```
typedef struct
{
    u32 flags;
    u32 main;
    u32 gsc;
    u32 alt;
    u32 io;
    u32 timeout_ms;
    u32 count;
} gsc_wait_t;
```

Fields	Description
flags	This is unused by wait cancel operations.
main	This specifies the set of GSC_WAIT_MAIN_* events whose wait requests are to be cancelled. Refer to section 4.7.48.2 on page 37.

<code>gsc</code>	This specifies the set of <code>GS_16AIO_WAIT_GSC_*</code> events whose wait requests are to be cancelled. Refer to section 4.7.48.3 on page 37.
<code>alt</code>	This is unused by the 16AIO driver and should be zero.
<code>io</code>	This specifies the set of <code>GS_16AIO_WAIT_IO_*</code> events whose wait requests are to be cancelled. Refer to section 4.7.48.4 on page 37.
<code>timeout_ms</code>	This is unused by wait cancel operations.
<code>count</code>	Upon return this indicates the number of waits that were cancelled.

4.7.48. GS_16AIO_IOCTL_WAIT_EVENT

This service blocks a thread until any one of a specified set of events occurs, or until a timeout lapses, whichever occurs first. The set of possible events to wait for are specified in the structure's `main`, `gsc`, `alt` and `io` fields. All field values must be valid and at least one event must be specified. If the thread is resumed because one of the referenced events has occurred, then the bit for the respective event is the only event bit that will be set. All other event bits and fields will be zero. (Multiple event bits will be set only if the events occur simultaneously.)

NOTE: The service waits only for the first of the specified events, not for all specified events.

NOTE: A wait timeout is reported via the `gsc_wait_t` structure's `flags` field having the `GSC_WAIT_FLAG_TIMEOUT` flag set, rather than via an `ETIMEDOUT` error.

Usage

Argument	Description
<code>request</code>	<code>GS_16AIO_IOCTL_WAIT_EVENT</code>
<code>arg</code>	<code>gsc_wait_t*</code>

Definition

```
typedef struct
{
    u32 flags;
    u32 main;
    u32 gsc;
    u32 alt;
    u32 io;
    u32 timeout_ms;
    u32 count;
} gsc_wait_t;
```

Fields	Description
<code>flags</code>	This must initially be zero. Upon return this indicates the reason that the thread was resumed. Refer to section 4.7.48.1 on page 37.
<code>main</code>	This specifies any number of <code>GSC_WAIT_MAIN_*</code> events that the thread is to wait for. Refer to section 4.7.48.2 on page 37.
<code>gsc</code>	This specifies any number of <code>GS_16AIO_WAIT_GSC_*</code> events that the thread is to wait for. Refer to section 4.7.48.3 on page 37.
<code>alt</code>	This is unused by the 16AIO driver and must be zero.
<code>io</code>	This specifies any number of <code>GS_16AIO_WAIT_IO_*</code> events that the thread is to wait for. Refer to section 4.7.48.4 on page 37.
<code>timeout_ms</code>	This specified the maximum amount of time, in milliseconds, that the thread is to wait for any of the referenced events. A value of zero means “do not timeout at all”. If non-zero, then upon return the value will be the approximate amount of time actually waited.
<code>count</code>	This is unused by wait event operations and must be zero.

4.7.48.1. `gsc_wait_t.flags` Options

Upon return from a wait request the wait structure's `flags` field will indicate the reason that the thread was resumed. Only one of the below options will be set.

Fields	Description
<code>GSC_WAIT_FLAG_CANCEL</code>	The wait request was cancelled.
<code>GSC_WAIT_FLAG_DONE</code>	One of the referenced events occurred.
<code>GSC_WAIT_FLAG_TIMEOUT</code>	The timeout period lapsed before a referenced event occurred.

4.7.48.2. `gsc_wait_t.main` Options

The wait structure's `main` field may specify any of the below primary interrupt options. These interrupt options are supported by the 16AIO and other General Standards products.

Fields	Description
<code>GSC_WAIT_MAIN_DMA0</code>	This refers to the DMA Done interrupt on DMA engine number zero.
<code>GSC_WAIT_MAIN_DMA1</code>	This refers to the DMA Done interrupt on DMA engine number one.
<code>GSC_WAIT_MAIN_GSC</code>	This refers to any of the Interrupt Control/Status Register interrupts.
<code>GSC_WAIT_MAIN_OTHER</code>	This generally refers to an interrupt generated by another device sharing the same interrupt as the 16AIO.
<code>GSC_WAIT_MAIN_PCI</code>	This refers to any interrupt generated by the 16AIO.
<code>GSC_WAIT_MAIN_SPURIOUS</code>	This refers to board interrupts which should never be generated.
<code>GSC_WAIT_MAIN_UNKNOWN</code>	This refers to board interrupts whose source could not be identified.

4.7.48.3. `gsc_wait_t.gsc` Options

The wait structure's `gsc` field may specify any combination of the below interrupt options. These are the interrupt options referenced in the Interrupt Control Register. Applications are responsible for selecting the desired interrupt options. Refer to `GS_16AIO_IOCTL_IRQ0_SEL` (section 4.7.28, page 27), `GS_16AIO_IOCTL_IRQ1_SEL` (section 4.7.29, page 28) and `GS_16AIO_IOCTL_IRQ2_SEL` (section 4.7.30, page 28).

Value	Description
<code>GS_16AIO_WAIT_GSC_AI_BUF_THR_H2L</code>	This refers to a high-to-low transition on the Analog Input Buffer Threshold Status.
<code>GS_16AIO_WAIT_GSC_AI_BUF_THR_L2H</code>	This refers to a low-to-high transition on the Analog Input Buffer Threshold Status.
<code>GS_16AIO_WAIT_GSC_AO_BUF_THR_H2L</code>	This refers to a high-to-low transition on the Analog Output Buffer Threshold Status.
<code>GS_16AIO_WAIT_GSC_AO_BUF_THR_L2H</code>	This refers to a low-to-high transition on the Analog Output Buffer Threshold Status.
<code>GS_16AIO_WAIT_GSC_AO_BURST_DONE</code>	This refers to the completion of Analog Output burst operation.
<code>GS_16AIO_WAIT_GSC_AUTOCAL_DONE</code>	This refers to the completion of an autocalibration cycle.
<code>GS_16AIO_WAIT_GSC_AUX_IN_H2L</code>	This refers to a high-to-low transition on the Auxiliary Input.
<code>GS_16AIO_WAIT_GSC_AUX_IN_L2H</code>	This refers to a low-to-high transition on the Auxiliary Input.
<code>GS_16AIO_WAIT_GSC_IDLE_INIT</code>	This refers to the completion of an initialization cycle.

4.7.48.4. `gsc_wait_t.io` Options

The wait structure's `io` field may specify any of the below event options. These events are generated in response to application board data read requests.

Fields	Description
GS_16AIO_WAIT_IO_RX_ABORT	This refers to read requests which have been aborted.
GS_16AIO_WAIT_IO_RX_DONE	This refers to read requests which have been satisfied.
GS_16AIO_WAIT_IO_RX_ERROR	This refers to read requests which end due to an error.
GS_16AIO_WAIT_IO_RX_TIMEOUT	This refers to read requests which end due to the timeout period lapse.
GS_16AIO_WAIT_IO_TX_ABORT	This refers to write requests which have been aborted.
GS_16AIO_WAIT_IO_TX_DONE	This refers to write requests which have been satisfied.
GS_16AIO_WAIT_IO_TX_ERROR	This refers to write requests which end due to an error.
GS_16AIO_WAIT_IO_TX_TIMEOUT	This refers to write requests which end due to the timeout period lapse.

4.7.49. GS_16AIO_IOCTL_WAIT_STATUS

This service counts all threads blocked via the GS_16AIO_IOCTL_WAIT_EVENT IOCTL service (section 4.7.48, page 36), according to the provided criteria. A match is made when a waiting thread's wait criteria matches any of the criteria specified in the structure passed to this service.

NOTE: The driver itself makes use of the wait services for various internal operations. Driver initiated waits are ignored by application status requests.

Usage

Argument	Description
request	GS_16AIO_IOCTL_WAIT_STATUS
arg	gsc_wait_t*

Definition

```
typedef struct
{
    u32 flags;
    u32 main;
    u32 gsc;
    u32 alt;
    u32 io;
    u32 timeout_ms;
    u32 count;
} gsc_wait_t;
```

Fields	Description
flags	This is unused by wait status operations.
main	This specifies the set of GSC_WAIT_MAIN_* events whose wait requests are to be counted. Refer to section 4.7.48.2 on page 37.
gsc	This specifies the set of GS_16AIO_WAIT_GSC_* events whose wait requests are to be counted. Refer to section 4.7.48.3 on page 37.
alt	This is unused by the 16AIO driver and should be zero.
io	This specifies the set of GS_16AIO_WAIT_IO_* events whose wait requests are to be counted. Refer to section 4.7.48.4 on page 37.
timeout_ms	This is unused by wait status operations.
Count	Upon return this indicates the number of waits that met any of the specified criteria.

5. The Driver

NOTE: Contact General Standards Corporation if additional driver functionality is required.

5.1. Files

The driver is built into an OS specific executable. The pertinent files are summarized in the table below.

Description	Files	Location	OS
Source Files	*.c, *.h	.../driver/	All
Header File	16aio.h	.../driver /	All
Driver File	16aio.ko †	.../driver/	Linux (kernels version 2.6 and later)
	16aio.o †	.../driver/	Linux (kernels version 2.4 and earlier)
	16aio.rta ‡	...\\driver\\	INtime

† The Linux run time executable is built into a loadable kernel module.

‡ The INtime run time executable is provided in the form of an INtime executable.

5.2. Build

For instructions on building the driver refer to this same section number in the OS specific 16AIO driver user manual.

5.3. Startup

For instructions on starting the driver executable refer to this same section number in the OS specific 16AIO driver user manual.

5.4. Verification

For specific instruction on verifying that the driver has been loaded and is running refer to this same section number in the OS specific 16AIO driver user manual.

5.5. Version

For instructions on obtaining the driver version number refer to this same section number in the OS specific 16AIO driver user manual.

5.6. Shutdown

For instructions on terminating the driver executable refer to this same section number in the OS specific 16AIO driver user manual.

6. Document Source Code Examples

The source code examples included in this document are built into a statically linkable library usable with console applications. The purpose of these files is to verify that the documentation samples compile and to provide a library of working sample code to assist in a user's learning curve and application development effort.

6.1. Files

The library files are summarized in the table below.

Description	Files	Location	OS
Source Files	*.c, *.h	.../docsrc/	All
Header File	16aio_dsl.h	.../include/	All
Library File	16aio_dsl.a	.../lib/	Linux
	16aio_dsl.lib	...\\lib\\	INtime

6.2. Build

For library build instructions refer to this same section number in the OS specific 16AIO driver user manual.

6.3. Library Use

For library usage information refer to this same section number in the OS specific 16AIO driver user manual.

7. Utilities Source Code

The API Library installation includes a body of utility source code designed to aid in the understanding and use of the interface calls and IOCTL services. Utility sources are also included for device independent and common, general-purpose services. Most of the utilities are implemented as visual wrappers around the corresponding services to facilitate structured console output for the sample applications. The utility sources are compiled and linked into static libraries to simplify their use. An additional purpose of these utility services is to provide a library of working sample code to assist in a user's learning curve and application development effort.

For each API function there is a corresponding utility source file with a corresponding utility service. As an example, for the API function `gs_16aio_open()` there is the utility file `open.c` containing the utility function `gs_16aio_open_util()`. The naming pattern is as follows: API function `gs_16aio_xxxx()`, utility file name `xxxx.c`, utility function `gs_16aio_xxxx_util()`. Additionally, for each IOCTL code there is a corresponding utility source file with a corresponding utility service. As an example, for IOCTL code `GS_16AIO_IOCTL_QUERY` there is the utility file `query.c` containing the utility function `gs_16aio_query()`. The naming pattern is as follows: IOCTL code `GS_16AIO_IOCTL_XXXX`, utility file name `xxxx.c`, utility function `gs_16aio_xxxx()`.

7.1. Files

The utility files are summarized in the table below.

Description	Files	Location	OS
Source Files	*.c, *.h, makefile/utils/	All
Header File	16aio_utils.h	.../include/	All
Library File	16aio_utils.a gsc_utils.a os_utils.a plx_utils.a	.../lib/	Linux
	16aio_utils.lib gsc_utils.lib os_utils.lib plx_utils.lib	...\\lib\\	INtime

7.2. Build

For library build instruction refer to this same section number in the OS specific 16AIO driver user manual.

7.3. Library Use

For library usage information refer to this same section number in the OS specific 16AIO driver user manual.

8. Operating Information

This section explains some basic operational procedures for using the 16AIO. This is in no way intended to be a comprehensive guide. This is simply to address a very few issues relating to their use. For additional operating information refer to this same section number in the OS specific *16AIO Driver User Manual*.

8.1. Debugging Aids

The driver package includes the following items useful for development and/or debugging aids.

8.1.1. Device Identification

When communicating with technical support complete device identification is virtually always necessary. The *id* example application is provided for this specific purpose. This is a text only console application. The output can be piped to a file, which can then be emailed to GSC technical support when requested. Locate the application as follows.

Description	File	Location	OS
Application	id	.../id/	Linux
	id.rta	...\\id\\	INtime

8.1.2. Detailed Register Dump

Among the utility services provided is a function to generate a detailed listing of device registers to the console. When used, the function is typically used to verify device configuration. In these cases, the function should be called after complete device configuration and before the first I/O call. When intended for sending to GSC tech support, please set the *detail* arguments to 1. The function arguments are as follows. The utility location is given in the subsequent table.

Argument	Description
fd	This is the file descriptor used to access the device.
detail	If non-zero the register dump will include details of each register field.

Description	File/Name	Location	OS
Function	gs_16aio_reg_list()	Source File	ALL
Source File	reg.c	.../utils/	ALL
Header File	16aio_utils.h	.../include/	ALL
Library File	16aio_utils.a	.../lib/	Linux
	16aio_utils.lib	...\\lib\\	INtime

8.2. Basic Analog Input Configuration

The basic steps for Analog Input configuration are illustrated in the utility function noted below. The table also gives the location of the source file, the header file and the corresponding library containing the executable code. The referenced files are included via the Main Header and Main Library.

Description	File/Name	Location	OS
Function	gs_16aio_config_ai()	Source File	ALL
Source File	config_ai.c	.../utils/	ALL
Header File	16aio_utils.h	.../include/	ALL

Library File	16aio_utils.a	.../lib/	Linux
	16aio_utils.lib	...\\lib\\	INtime

8.3. Basic Analog Output Configuration

The basic steps for Analog Output configuration are illustrated in the utility function noted below. The table also gives the location of the source file, the header file and the corresponding library containing the executable code. The referenced files are included via the Main Header and Main Library.

Description	File/Name	Location	OS
Function	gs_16aio_config_ao()	Source File	ALL
Source File	config_ao.c	.../utils/	ALL
Header File	16aio_utils.h	.../include/	ALL
Library File	16aio_utils.a	.../lib/	Linux
	16aio_utils.lib	...\\lib\\	INtime

8.4. Data Transfer Modes

All I/O requests move data between the board's FIFO buffers, intermediate driver buffers, and application memory buffers. The data is processed in chunks no larger than the size of the FIFOs. The process used to move the data between the FIFOs and the intermediate buffers is according to the I/O mode selection.

8.4.1. PIO - Programmed I/O

This method transfers data through repetitive register accesses. While this method is not very efficient it is the most reliable method and the only method that should be used with an I/O timeout value of zero.

8.4.2. DMA - Block Mode DMA

For Block Mode DMA the driver initiates DMA transfers only after a sufficient volume of data or space has become available to accommodate the transfer. For read requests the volume is sufficient only when the fill level exceeds the threshold level. When the threshold level is insufficient the driver waits for 1ms before rechecking the fill level. For write requests the volume is sufficient when the fill level is below the threshold. When the threshold level is insufficient the driver waits for 1ms before rechecking the fill level. Once the fill level is sufficient the driver initiates a DMA transfer then sleeps until the DMA Done interrupt is received. Using this DMA mode, a user request typically consists of numerous smaller individual DMA transfers.

9. Sample Applications

For information on the sample applications refer to this same section number in the OS specific 16AIO driver user manual.

NOTE: Most of the sample applications are available in each driver release. Some are OS specific and are therefore included only with the appropriate OS specific 16AIO driver release.

Document History

Revision	Description
March 17, 2026	Updated to driver release version 6.0.117.x.x. Replaced all instances the “aio_” prefix with “gs_16aio_”, both lower and upper case. Removed the “util_” prefix from the utility source files.
August 12, 2024	Updated to driver release version 5.8.111.x.x.
June 22, 2023	Updated to driver release version 5.7.104.x.x. Added voltage range information for boards with the -16HV60V ordering option. Updated the description of the Autocalibration service. Numerous, minor editorial changes. Renamed all forms of Auto_Calibrate instances to autocal. Renamed all forms of auto_Cal_Sts instances to Autocal_Status.
October 7, 2022	Updated to driver release version 5.6.101.x.x. Updated the information for the open and close calls.
July 8, 2022	Updated to driver release version 5.6.100.x.x. Minor editorial changes.
February 15, 2022	Updated to driver release version 5.5.96.x.0. Clarified the default state of the digital I/O ports. Clarified operation of the aio_open() call. Clarified operation of the aio_close() call.
January 8, 2021	Updated to driver release version 5.4.92.x.0. Numerous editorial changes. Added WAIT_EVENT note.
July 16, 2019	Updated to driver release version 5.4.86.x.0. Minor editorial changes. Added a licensing subsection.
May 7, 2019	Updated to driver release version 5.3.85.x.0. Overhauled document.
November 6, 2018	Updated to driver release version 5.3.81.x.0. Updated Block Mode DMA macro and associated information. Renamed GSC_WAIT_IO_*** macros to AIO_WAIT_IO_***.
July 11, 2018	Updated to driver release version 5.3.79.x.0.
July 8, 2018	Initial release, version 5.2.77.x.0.