

16AO16

16-Bit, 16 Channel High-Speed Analog Output Board

**All Form Factors
...-16AO16/FLV**

Linux Device Driver And API Library User Manual

**Manual Revision: February 6, 2025
Driver Release Version 3.12.111.50.0**

**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 © 2003-2025, **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.

Table of Contents

1. Introduction.....	8
1.1. Purpose	8
1.2. Acronyms.....	8
1.3. Definitions.....	8
1.4. Software Overview.....	8
1.4.1. Basic Software Architecture	8
1.4.2. API Library.....	9
1.4.3. Device Driver	9
1.5. Hardware Overview.....	9
1.6. Reference Material.....	9
1.7. Licensing.....	10
2. Installation.....	11
2.1. CPU and Kernel Support	11
2.1.1. 32-bit Support Under 64-bit Environments	12
2.2. The /proc/ File System	12
2.3. File List	12
2.4. Directory Structure.....	12
2.5. Installation.....	13
2.6. Removal	13
2.7. Overall Make Script	13
2.8. Environment Variables	14
2.8.1. GSC_API_COMP_FLAGS	14
2.8.2. GSC_API_LINK_FLAGS	14
2.8.3. GSC_LIB_COMP_FLAGS	14
2.8.4. GSC_LIB_LINK_FLAGS	15
2.8.5. GSC_APP_COMP_FLAGS	15
2.8.6. GSC_APP_LINK_FLAGS	15
3. Main Interface Files.....	16
3.1. Main Header File	16
3.2. Main Library File	16
3.2.1. Build	16
3.2.2. System Libraries.....	17
3.2.3. Shared Object Script: Build the Main Libraries as Shared Object Files	17
4. API Library	18
4.1. Files	18
4.2. Build	18
4.3. Library Use.....	18
4.4. Macros.....	18

4.4.1. IOCTL Services 19

4.4.2. Registers 19

4.5. Data Types 19

4.6. Functions..... 19

4.6.1. ao16_close() 20

4.6.2. ao16_init() 20

4.6.3. ao16_ioctl() 21

4.6.4. ao16_open()..... 22

4.6.5. ao16_read()..... 23

4.6.6. ao16_write() 24

4.7. IOCTL Services..... 25

4.7.1. AO16_IOCTL_AUTOCAL..... 25

4.7.2. AO16_IOCTL_AUTOCAL_STATUS..... 25

4.7.3. AO16_IOCTL_BUFFER_CLEAR..... 25

4.7.4. AO16_IOCTL_BUFFER_MODE..... 26

4.7.5. AO16_IOCTL_BUFFER_OVER_DATA..... 26

4.7.6. AO16_IOCTL_BUFFER_OVER_FRAME..... 26

4.7.7. AO16_IOCTL_BUFFER_SIZE..... 27

4.7.8. AO16_IOCTL_BUFFER_STATUS..... 27

4.7.9. AO16_IOCTL_BURST_ENABLE..... 28

4.7.10. AO16_IOCTL_BURST_READY..... 28

4.7.11. AO16_IOCTL_BURST_TRIG_SRC..... 28

4.7.12. AO16_IOCTL_BURST_TRIGGER..... 29

4.7.13. AO16_IOCTL_CBL_ISO_CLOCK_IO..... 29

4.7.14. AO16_IOCTL_CBL_ISO_DAC_CLK_OUT..... 29

4.7.15. AO16_IOCTL_CBL_ISO_TRIG_OUT..... 29

4.7.16. AO16_IOCTL_CBL_POL_CLOCK_IO..... 30

4.7.17. AO16_IOCTL_CBL_POL_DAC_CLK_OUT..... 30

4.7.18. AO16_IOCTL_CBL_POL_TRIG_IN..... 30

4.7.19. AO16_IOCTL_CBL_POL_TRIG_OUT..... 31

4.7.20. AO16_IOCTL_CHANNEL_SEL..... 31

4.7.21. AO16_IOCTL_CLOCK_ENABLE..... 31

4.7.22. AO16_IOCTL_CLOCK_READY..... 32

4.7.23. AO16_IOCTL_CLOCK_REF_SRC..... 32

4.7.24. AO16_IOCTL_CLOCK_SRC..... 32

4.7.25. AO16_IOCTL_CLOCK_SW..... 33

4.7.26. AO16_IOCTL_DATA_FORMAT..... 33

4.7.27. AO16_IOCTL_GROUND_SENSE..... 33

4.7.28. AO16_IOCTL_INITIALIZE..... 33

4.7.29. AO16_IOCTL_IRQ_SEL..... 34

4.7.30. AO16_IOCTL_LOAD_READY..... 34

4.7.31. AO16_IOCTL_LOAD_REQUEST..... 35

4.7.32. AO16_IOCTL_NCLK..... 35

4.7.33. AO16_IOCTL_NRATE..... 35

4.7.34. AO16_IOCTL_OUTPUT_FILTER..... 35

4.7.35. AO16_IOCTL_OUTPUT_MODE..... 36

4.7.36. AO16_IOCTL_QUERY..... 36

4.7.37. AO16_IOCTL_RANGE..... 38

4.7.38. AO16_IOCTL_REG_MOD..... 39

4.7.39. AO16_IOCTL_REG_READ..... 39

4.7.40. AO16_IOCTL_REG_WRITE..... 40

4.7.41. AO16_IOCTL_TX_IO_ABORT..... 40

4.7.42. AO16_IOCTL_TX_IO_MODE..... 40

4.7.43. AO16_IOCTL_TX_IO_OVER_DATA..... 41

4.7.44. AO16_IOCTL_TX_IO_OVER_FRAME.....	41
4.7.45. AO16_IOCTL_TX_IO_TIMEOUT	42
4.7.46. AO16_IOCTL_WAIT_CANCEL	42
4.7.47. AO16_IOCTL_WAIT_EVENT	43
4.7.48. AO16_IOCTL_WAIT_STATUS	45
4.7.49. AO16_IOCTL_WATCHDOG_ENABLE.....	45
4.7.50. AO16_IOCTL_WATCHDOG_OUTPUT.....	46
4.7.51. AO16_IOCTL_XCVR_TYPE.....	46
5. The Driver.....	47
5.1. Files	47
5.2. Build	47
5.3. Startup	47
5.3.1. Manual Driver Startup Procedures	47
5.3.2. Automatic Driver Startup Procedures.....	48
5.4. Verification	50
5.5. Version	50
5.6. Shutdown	50
6. Document Source Code Examples.....	51
6.1. Files	51
6.2. Build	51
6.3. Library Use.....	51
7. Utilities Source Code.....	52
7.1. Files	52
7.2. Build	52
7.3. Library Use.....	52
8. Operating Information	53
8.1. Debugging Aids	53
8.1.1. Device Identification	53
8.1.2. Detailed Register Dump	53
8.2. Analog Output Configuration.....	53
8.3. Data Transfer Modes	53
8.3.1. PIO - Programmed I/O	54
8.3.2. BMDMA - Block Mode DMA	54
8.3.3. DMDMA - Demand Mode DMA	54
8.4. Output Overflow Errors.....	54
8.4.1. Data Overflow	54
8.4.2. Frame Overflow.....	54
8.4.3. Recovery Action.....	55
9. Sample Applications	56
9.1. aout - Analog Output - ../aout/	56

9.2. clockout - Clock Output - .../clockout/.....56
9.3. fsamp - Sample Rate - .../fsamp/56
9.4. id - Identify Board - .../id/56
9.5. mcao - Multi-Channel Analog Output - .../mcao/.....56
9.6. regs - Register Access - .../regs/56
9.7. sbtest - Single Board Test - .../sbtest/56
9.8. txrate - Transmit Rate - .../txrate/56
Document History 57

Table of Figures

Figure 1 Basic architectural representation.....9

1. Introduction

1.1. Purpose

The purpose of this document is to describe the interface to the 16AO16 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 16AO16 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
API	Application Programming Interface
BMDMA	Block Mode DMA
DAC	Digital-to-Analog Converter
DMA	Direct Memory Access
DMDMA	Demand Mode DMA
GSC	General Standards Corporation
PCI	Peripheral Component Interconnect
PCIe	PCI Express
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 16AO16 installation directory or any of its subdirectories.
16AO16	This is used as a general reference to any device supported by this driver.
API Library	This is a library that provides application-level access to 16AO16 hardware.
Application	This is a user mode process, which runs in user space with user mode privileges.
Driver	This is the 16AO16 device driver, which runs in kernel space with kernel mode privileges.
Library	This is usually a general reference to the API Library.

1.4. Software Overview

1.4.1. Basic Software Architecture

This section describes the general architecture for the basic components that comprise 16AO16 applications. The overall architecture is illustrated in Figure 1 below.

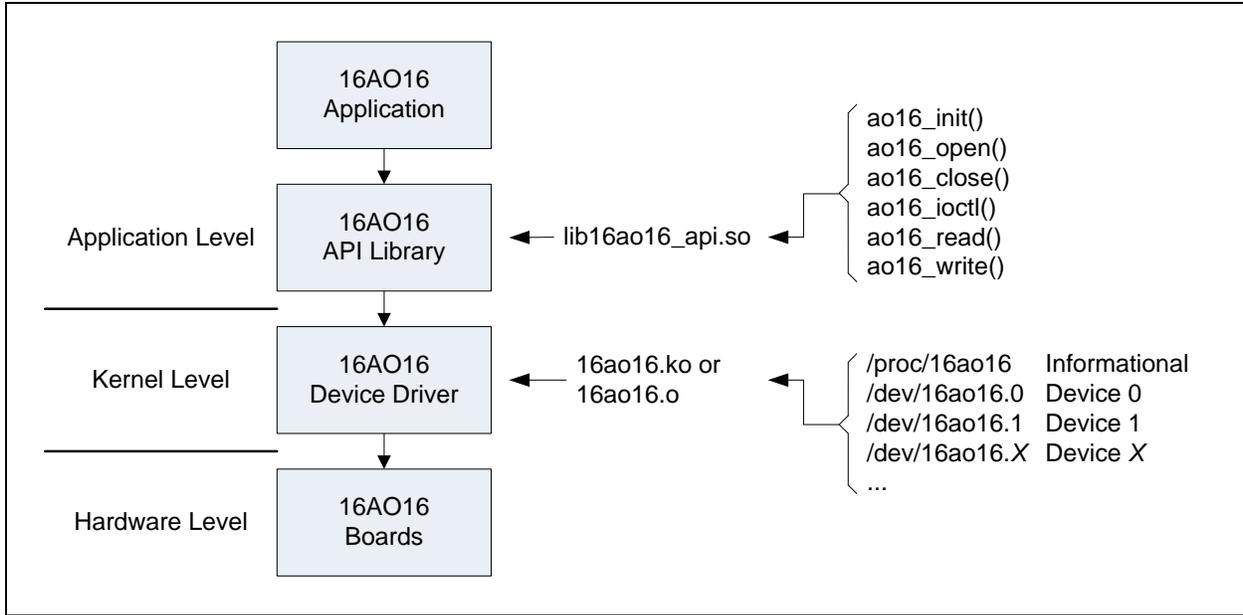


Figure 1 Basic architectural representation.

1.4.2. API Library

The primary means of accessing 16AO16 boards is via the 16AO16 API Library. This library forms a layer between the application and the driver. Additional information is given in section 4 (page 18). With the library, applications are able to open and close a device and, while open, perform I/O control and read and write operations.

1.4.3. Device Driver

The device driver is the host software that provides a means of communicating directly with 16AO16 hardware. The driver executes under control of the operating system and runs in Kernel Mode as a Kernel Mode device driver. The driver is implemented as a standard dynamically loadable Linux device driver written in the C programming language. While applications can access the driver directly without use of the API Library, it is recommended that all access is made through the library.

1.5. Hardware Overview

The 16AO16 is a 16-channel high speed analog output board. Each channel has a dedicated D/A converter that provides 16-bits of resolution at up to 450K samples per second. Working together the active channels can operate at an aggregate rate of up to 7.2M samples per second. An on-board FIFO of 256K samples buffers streaming data between the PCI interface and the cable interface. The FIFO can also be used for pattern generation by recycling data written to the FIFO. The PCI interface is compliant with PCI Revision 2.3, operates at up to 66MHz and supports universal signaling (3.3V or 5V). The cable interface output voltage range is selectable as ± 10 Volts, ± 5 Volts, ± 2.5 Volts or ± 1.25 Volts. The analog outputs support 3-wire balanced differential operation or, optionally, 2-wire single ended operation. The clocking source can be either the on-board clock or an external clock. The board also supports multi-board synchronization.

1.6. Reference Material

The following reference material may be of particular benefit in using the 16AO16. The specifications provide the information necessary for an in depth understanding of the specialized features implemented on this board.

- The applicable *16AO16 User Manual* from General Standards Corporation.

- The *PCI9056 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.

2. Installation

2.1. CPU and Kernel Support

The driver is designed to operate with Linux kernel versions 6.x, 5.x, 4.x, 3.x, 2.6, 2.4 and 2.2 running on a PC system with one or more x86 processors. This release of the driver supports the below listed kernels.

Kernel	Distribution
6.8.5	Red Hat Fedora Core 40
6.5.6	Red Hat Fedora Core 39
6.2.9	Red Hat Fedora Core 38
6.0.7	Red Hat Fedora Core 37
5.17.5	Red Hat Fedora Core 36
5.14.10	Red Hat Fedora Core 35
5.11.12	Red Hat Fedora Core 34
5.8.15	Red Hat Fedora Core 33
5.6.6	Red Hat Fedora Core 32
5.3.7	Red Hat Fedora Core 31
5.0.9	Red Hat Fedora Core 30
4.18.16	Red Hat Fedora Core 29
4.16.3	Red Hat Fedora Core 28
4.13.9	Red Hat Fedora Core 27
4.11.8	Red Hat Fedora Core 26
4.8.6	Red Hat Fedora Core 25
4.5.5	Red Hat Fedora Core 24
4.2.3	Red Hat Fedora Core 23
4.0.4	Red Hat Fedora Core 22
3.17.4	Red Hat Fedora Core 21
3.11.10	Red Hat Fedora Core 20
3.9.5	Red Hat Fedora Core 19
3.6.10	Red Hat Fedora Core 18
3.3.4	Red Hat Fedora Core 17
3.1.0	Red Hat Fedora Core 16
2.6.38	Red Hat Fedora Core 15
2.6.35	Red Hat Fedora Core 14
2.6.33	Red Hat Fedora Core 13
2.6.31	Red Hat Fedora Core 12
2.6.29	Red Hat Fedora Core 11
2.6.27	Red Hat Fedora Core 10
2.6.25	Red Hat Fedora Core 9
2.6.23	Red Hat Fedora Core 8
2.6.21	Red Hat Fedora Core 7
2.6.18	Red Hat Fedora Core 6
2.6.15	Red Hat Fedora Core 5
2.6.11	Red Hat Fedora Core 4
2.6.9	Red Hat Fedora Core 3
2.4.18	Red Hat 8.0

NOTE: Some older kernel versions are supported (the sources are maintained), but are not tested.

NOTE: While only Red Hat Fedora distributions are listed, numerous other distributions are supported and have been tested on an as needed basis.

NOTE: The driver will have to be built before being used as it is provided in source form only.

NOTE: The driver has not been tested with a non-versioned kernel.

NOTE: The driver is designed for SMP support, but has not undergone SMP specific testing.

2.1.1. 32-bit Support Under 64-bit Environments

This driver supports 32-bit applications under 64-bit environments. The availability of this feature in the kernel depends on a 64-bit kernel being configured to support 32-bit application compatibility. Additionally, 2.6 kernels prior to 2.6.11 implemented 32-bit compatibility in a way that resulted in some drivers not being able to take advantage of the feature. (In these kernels a driver’s IOCTL command codes must be globally unique. Beginning with 2.6.11 this requirement has been lifted.) If the driver is not able to provide 32-bit support under a 64-bit kernel, the “32-bit support” field in the `/proc/16ao16` file will be “no”.

2.2. The /proc/ File System

While the driver is running, the text file `/proc/16ao16` can be read to obtain information about the driver and the boards it detects. Each file entry includes an entry name followed immediately by a colon, a space character, and the entry value. Below is an example of what appears in the file, followed by descriptions of each entry.

```
version: 3.12.111.50
32-bit support: yes
boards: 1
models: 16AO16
```

Entry	Description
version	This gives the driver version number in the form <code>x.x.x.x</code> .
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 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.

2.3. File List

This release consists of the below listed primary files. The archive content is described in following subsections.

File	Description
<code>16ao16.linux.tar.gz</code>	This archive contains the driver, the API Library and all related files.
<code>16ao16_linux_um.pdf</code>	This is a PDF version of this user manual, which is included in the archive.

2.4. Directory Structure

The following table describes the directory structure utilized by the installed files. During installation the directory structure is created and populated with the respective files.

Directory	Description
<code>16ao16/</code>	This is the driver root directory. It contains the documentation, the Overall Make Script (section 2.7, page 13) and the below listed subdirectories.
<code>.../api/</code>	This directory contains the API Library source files (section 3.2.3, page 17).
<code>.../docsrc/</code>	This directory contains the source files for the code samples given in this document (section 6, page 51).

.../driver/	This directory contains the device driver source files (section 5, page 47).
.../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 56).
.../utils/	This directory contains the source files for the utility libraries used by the sample applications (section 7, page 52).

2.5. Installation

Perform installation following the below listed steps. This installs the device driver, the API Library and all related sources and documentation.

1. Create and change to the directory where the files are to be installed, such as `/usr/src/linux/drivers/`. (The path name may vary among distributions and kernel versions.)
2. Copy the archive file `16ao16.linux.tar.gz` into the current directory.
3. Issue the following command to decompress and extract the files from the provided archive. This creates the directory `16ao16` in the current directory, and then copies all of the archive's files into this new directory.

```
tar -xzvf 16ao16.linux.tar.gz
```

2.6. Removal

Perform removal following the below listed steps. This removes the device driver, the API Library and all related sources and documentation.

NOTE: The following steps may require elevated privileges.

1. Shutdown the driver as described in section 5.6 (page 50).
2. Change to the directory where the driver archive was installed, which may have been `/usr/src/linux/drivers/`. (The path name may vary among distributions and kernel versions.)
3. Issue the below command to remove the driver archive and all of the installed driver files.

```
rm -rf 16ao16.linux.tar.gz 16ao16
```

4. Issue the below command to remove all of the installed device nodes.

```
rm -f /dev/16ao16.*
```

5. If the automatic startup procedure was adopted (section 5.3.2, page 48), then edit the system startup script `rc.local` and remove the line that invokes the 16AO16's start script. The file `rc.local` should be located in the `/etc/rc.d/` directory.

2.7. Overall Make Script

An Overall Make Script is included in the root installation directory. Executing this script will perform a make for all build targets included in the release. The script also loads the driver and copies the API Library to `/usr/lib/`. The script is named `make_all`. Follow the below steps to perform an overall make and to load the driver.

NOTE: The following steps may require elevated privileges.

1. Change to the driver root directory (.../16ao16/).
2. Remove existing build targets using the below command. This does not unload the driver.

```
./make_all clean
```

3. Issue the following command to make all archive targets and to load the driver.

```
./make_all
```

2.8. Environment Variables

Some build environments may require compiler or linker options not present in the provided make files. To accommodate local environment specific requirements, the provided make files incorporate support for the following set of GSC specific environment variables.

2.8.1. GSC_API_COMP_FLAGS

This environment variable accommodates adding compiler command line options when compiling source files for the API Library. The compiler used by the API Library make file is “gcc”. The content of this environment variable is noted in the make file’s output to the screen. The table below shows a portion of the screen output. The “xxx” in the table refers to the contents of the environment variable. This environment variable has no effect on compiling any other distributed source files or linking of any object files.

Undefined or Empty	== Compiling: init.c == Compiling: ioctl.c == Compiling: open.c
Defined and Not Empty	== Compiling: init.c (added 'xxx') == Compiling: ioctl.c (added 'xxx') == Compiling: open.c (added 'xxx')

2.8.2. GSC_API_LINK_FLAGS

This environment variable accommodates adding linker command line options when linking object files for the API Library. The linker used by the API Library make file is “ld”. The content of this environment variable is noted in the make file’s output to the screen. The table below shows a portion of the screen output. The “xxx” in the table refers to the contents of the environment variable. This environment variable has no effect on compiling of any source files or linking of any other object files.

Undefined or Empty	==== Linking: ../lib/lib16ao16_api.so
Defined and Not Empty	==== Linking: ../lib/lib16ao16_api.so (added 'xxx')

2.8.3. GSC_LIB_COMP_FLAGS

This environment variable accommodates adding compiler command line options when compiling source files for the utility libraries. The compiler used by the utility library make files is “gcc”. The content of this environment variable is noted in the make files’ output to the screen. The table below shows a portion of the screen output. The “xxx” in the table refers to the contents of the environment variable. This environment variable has no effect on compiling any other distributed source files or linking of any object files.

Undefined or Empty	== Compiling: close.c == Compiling: init.c == Compiling: ioctl.c
Defined and Not Empty	== Compiling: close.c (added 'xxx') == Compiling: init.c (added 'xxx') == Compiling: ioctl.c (added 'xxx')

2.8.4. GSC_LIB_LINK_FLAGS

This environment variable accommodates adding linker command line options when linking object files for the utility libraries. The linker used by the utility library make files is “ld”. The content of this environment variable is noted in the make files’ output to the screen. The table below shows a portion of the screen output. The “xxx” in the table refers to the contents of the environment variable. This environment variable has no effect on compiling of any source files or linking of any other object files.

Undefined or Empty	==== Linking: ../lib/16aol6_utils.a
Defined and Not Empty	==== Linking: ../lib/16aol6_utils.a (added 'xxx')

2.8.5. GSC_APP_COMP_FLAGS

This environment variable accommodates adding compiler command line options when compiling source files for the sample applications. The compiler used by the sample application make files is “gcc”. The content of this environment variable is noted in the make files’ output to the screen. The table below shows a portion of the screen output. The “xxx” in the table refers to the contents of the environment variable. This environment variable has no effect on compiling any other distributed source files or linking of any object files.

Undefined or Empty	== Compiling: main.c == Compiling: perform.c
Defined and Not Empty	== Compiling: main.c (added 'xxx') == Compiling: perform.c (added 'xxx')

2.8.6. GSC_APP_LINK_FLAGS

This environment variable accommodates adding linker command line options when linking object files for the sample applications. The linker used by the sample application make files is “gcc”. The content of this environment variable is noted in the make files’ output to the screen. The table below shows a portion of the screen output. The “xxx” in the table refers to the contents of the environment variable. This environment variable has no effect on compiling of any source files or linking of any other object files.

Undefined or Empty	==== Linking: id
Defined and Not Empty	==== Linking: id (added 'xxx')

3. Main Interface Files

This section gives general information on the suggested device interface files to use when developing 16AO16 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 16AO16 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 16AO16 specific header files. Therefore, sources may include only this one 16AO16 header and make files may reference only this one 16AO16 include directory.

Description	File	Location
Header File	16ao16_main.h	.../include/

3.2. Main Library File

Throughout the remainder of this document references are made to various statically linkable libraries included as part of the 16AO16 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 16AO16 static library and only this one 16AO16 library directory.

Description	File	Location
Static Library	16ao16_main.a	.../lib/
	16ao16_multi.a	

NOTE: For applications using the 16AO16 and no other GSC devices, link the 16ao16_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 16AO16 API Library is implemented as a shared library and is thus not linked with the 16AO16 Main Library. The API Library must be linked with applications by adding the argument `-l16ao16_api` to the linker command line.

3.2.1. Build

The main library is built via the Overall Make Script (section 2.7, page 13). However, the main library can be built separately following the below steps.

1. Change to the directory where the main library resides (.../lib/).
2. Remove existing build targets using the below command.

```
make clean
```

3. Build the main library by issuing the below command.

```
make
```

3.2.2. System Libraries

In addition to linking the static library named above, as well as the API Library shared object file, applications may need to also link in additional system libraries as noted below.

Library	gcc Link Flag
Math	-lm
POSIX Thread	-lpthread
Real Time	-lrt

3.2.3. Shared Object Script: Build the Main Libraries as Shared Object Files

The main libraries built via the Overall Make Script (section 2.7, page 13) are static library files. Some applications however, require that the Main Libraries be accessed as shared object files. Generating shared object files require that all of the static libraries be recompiled for this purpose and linked as .so files. This is done using the Shared Object Script named below. When run, the script invokes the Overall Make Script to clean all existing build targets, deletes the two shared object files named below, if they exist, defines an environment variable used by all of the static library make files, invokes the Overall Make Script again to rebuild all existing build targets then invokes make on the library make file (.../lib/makefile) to link the shared object files. The required manual steps are as follows.

1. Change to the directory where the main library files reside (.../lib/).
2. Execute the below script.

```
./static_to_shared.sh
```

Running the above-named Shared Object Script produces the files given in the table below. These shared object files fulfill the same purpose as the similarly named static libraries as described in the note under section 3.2 above. Refer to that note when selecting which shared object file to use.

Description	File	Location
Shared Object Files	lib16ao16_main.so	.../lib/
	lib16ao16_multi.so	
	lib16ao16_all.so †	

† This library includes all generated libraries, including the API Library shared object file content.

The shared object files can be linked via two different methods. In the first method, the application linker command line can explicitly name the file in the same manner as is done were it a static library. This is the method used by the sample applications, all of which use the 16AO16 API Library, which itself is a shared object file. This file is also found in the .../lib/ subdirectory. In the second method, the .so files are copied to the /usr/lib/ subdirectory and are referenced on the application's linker command line as given in the table below.

Library	gcc Link Flag
lib16ao16_main.so	-l16ao16_main
lib16ao16_multi.so	-l16ao16_multi
lib16ao16_all.so †	-l16ao16_all

† This library includes all generated libraries, including the API Library shared object file content.

4. API Library

The 16AO16 API Library is the software interface between user applications and the 16AO16 device driver. The interface is accessed by including the header file `16ao16_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	File	Location
Source Files	*.c, *.h...	.../api/
Header File	16ao16_api.h	.../include/
Library File	lib16ao16_api.so	.../lib/ /usr/lib/ †

† The shared object library is automatically copied to `/usr/lib/` when it is built.

4.2. Build

The API Library is built via the Overall Make Script (section 2.7, page 13), but can be built separately following the below steps.

NOTE: The following steps may require elevated privileges.

1. Change to the directory where the library sources are installed (`.../api/`).
2. Remove existing build targets using the below command.

```
make clean
```

3. Compile the source files and build the library by issuing the below command. This step copies the API Library file to `/usr/lib/`.

```
make
```

4.3. Library Use

The library is used at application compile time, at application link time and at application run time. At compile time include the below listed header file in each source file using a component of the Library interface. Also, edit the include file search path to locate the header file in the below listed directory. At link time the Library's shared object file is linked via the linker command line. This can be done by naming the `.so` file explicitly or by adding the below linker command line argument. At run time the library is found in the directory `/usr/lib/`. (The shared object file is automatically copied to `/usr/lib/` when it is built.)

Description	File	Location	Linker Argument
Header File	16ao16_api.h	.../include/	
Shared Object Library	lib16ao16_api.so	.../lib/	
		/usr/lib/	-l16ao16_api

4.4. Macros

The API Library and driver interfaces include the following macros, which are defined in `16ao16.h`.

4.4.1. IOCTL Services

The IOCTL macros are documented in section 4.7 (page 25).

4.4.2. Registers

The following gives the complete set of 16AO16 registers.

4.4.2.1. GSC Registers

The following table gives the complete set of GSC specific 16AO16 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 *16AO16 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
AO16_GSC_ACR	Adjustable Clock Register (ACR)
AO16_GSC_AVR	Autocalibration Values Register (AVR)
AO16_GSC_BCR	Board Control Register (BCR)
AO16_GSC_BOR	Buffer Operations Register (BOR)
AO16_GSC_CSR	Channel Selection Register (CSR)
AO16_GSC_FOR	Firmware Options Register (FOR)
AO16_GSC_ODBR	Output Data Buffer Register (ODBR)
AO16_GSC_SRR	Sample Rate Register (SRR)

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_pci9056.h`, which is automatically included via `16ao16_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_pci9056.h`, which is automatically included via `16ao16_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 25).

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
< 0	This is the value “(-errno)” (see errno.h).

4.6.1. ao16_close()

This function is the entry point to close a connection made via the API's open call (section 4.6.4, page 22). The device is put in an initialized state before this call returns.

Prototype

```
int ao16_close(int fd);
```

Argument	Description
fd	This is the file descriptor obtained from the open service (section 4.6.4, page 22).

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

Example

```
#include <stdio.h>

#include "16ao16_dsl.h"

int ao16_close_dsl(int fd)
{
    int errs;
    int ret;

    ret = ao16_close(fd);

    if (ret)
        printf("ERROR: ao16_close() returned %d\n", ret);

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

4.6.2. ao16_init()

This function is the entry point to initializing the 16AO16 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 ao16_init(void);
```

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

Example

```
#include <stdio.h>

#include "16ao16_dsl.h"

int ao16_init_dsl(void)
{
    int errs;
    int ret;

    ret = ao16_init();

    if (ret)
        printf("ERROR: ao16_init() returned %d\n", ret);

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

4.6.3. ao16_ioctl()

This function is the entry point to performing setup and control operations on a 16AO16. 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 25).

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

Prototype

```
int ao16_ioctl(int fd, int request, void* arg);
```

Argument	Description
fd	This is the file descriptor obtained from the open service (section 4.6.4, page 22).
request	This specifies the desired operation to be performed (section 4.7, page 25).
arg	This is specific to the IOCTL operation specified by the request argument.

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

Example

```
#include <stdio.h>

#include "16ao16_dsl.h"

int ao16_ioctl_dsl(int fd, int request, void* arg)
{
    int errs;
    int ret;

    ret = ao16_ioctl(fd, request, arg);

    if (ret)
```

```

        printf("ERROR: ao16_ioctl() returned %d\n", ret);

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

```

4.6.4. ao16_open()

This function is the entry point to open a connection to a 16AO16 board. Before returning, the initialize IOCTL service is called to reset all hardware and software settings to their defaults.

Prototype

```
int ao16_open(int device, int share, int* fd);
```

Argument	Description						
device	This is the zero-based index of the 16AO16 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" data-bbox="451 852 1265 951"> <thead> <tr> <th>Value</th> <th>Description</th> </tr> </thead> <tbody> <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> </tbody> </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 12).

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

Example

```

#include <stdio.h>

#include "16ao16_dsl.h"

int ao16_open_dsl(int device, int share, int* fd)
{
    int errs;
    int ret;

    ret = ao16_open(device, share, fd);

    if (ret)
        printf("ERROR: ao16_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. ao16_read()

This function is the entry point to reading data from an open connection. The function reads up to `bytes` bytes from the connection. The read service has no functionality for reading from 16AO16 devices. It is provided for informational purposes only via device index `-1` as read requests will acquire information from the driver (section 2.2, page 12) rather than data from a device.

NOTE: Attempts to read from 16AO16 devices will return an error.

Prototype

```
int ao16_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 22).
<code>dst</code>	The data read is put here.
<code>bytes</code>	This is the desired number of bytes to read.

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

Example

```
#include <stdio.h>

#include "16ao16_dsl.h"

int ao16_read_dsl(int fd, void* dst, size_t bytes, size_t* qty)
{
    int errs;
    int ret;

    ret = ao16_read(fd, dst, bytes);

    if (ret < 0)
        printf("ERROR: ao16_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. ao16_write()

This function is the entry point to writing data to an open 16AO16. This function should only be called after a successful open of the respective device. The function writes up to `bytes` bytes to the board. Upon entry to this service the driver checks for output buffer data overflow and frame overflow errors. If either error status has been asserted, then the service immediately returns `-5`, which is `-EIO` from the system header `errno.h`. See section 8.4 (page 54) for additional information.

NOTE: Write requests are not supported for an open on device index `-1`.

Prototype

```
int ao16_write(int fd, const void* src, size_t bytes);
```

Argument	Description
<code>fd</code>	This is the file descriptor obtained from the open service (section 4.6.4, page 22).
<code>src</code>	The data written comes from here.
<code>bytes</code>	This is the desired number of bytes to write. This must be a multiple of four (4).

Return Value	Description
0 to <code>bytes</code>	The operation succeeded. A value less than <code>bytes</code> indicates that the I/O timeout period lapsed (section 4.7.45, page 42) before the entire request could be satisfied.
< 0	An error occurred. See error value description above.

Example

```

#include <stdio.h>

#include "16ao16_dsl.h"

int ao16_write_dsl(int fd, const void* src, size_t bytes, size_t*
qty)
{
    int errs;
    int ret;

    ret = ao16_write(fd, src, bytes);

    if (ret < 0)
        printf("ERROR: ao16_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 16AO16 API Library and device driver implement the following IOCTL services. Each service is described along with the applicable `ao16_ioctl()` function arguments.

4.7.1. AO16_IOCTL_AUTOCAL

This service initiates an autocalibration cycle. Most configuration setting should be made before running an autocalibration cycle. The driver waits for the operation to complete before returning.

NOTE: The autocalibration cycle for the 16AO16FLV takes noticeably longer than for the basic 16AO16; 20 seconds vs 5. However, both periods are reduced somewhat for 12 channel and eight channel boards.

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	AO16_IOCTL_AUTOCAL
arg	Not used.

4.7.2. AO16_IOCTL_AUTOCAL_STATUS

This service retrieves the autocalibration completion status.

Usage

Argument	Description
Request	AO16_IOCTL_AUTOCAL_STATUS
arg	s32*

The value returned will be one of the following.

Value	Description
AO16_AUTOCAL_STATUS_ACTIVE	Autocalibration is in progress.
AO16_AUTOCAL_STATUS_FAIL	Autocalibration failed.
AO16_AUTOCAL_STATUS_PASS	Autocalibration passed.

4.7.3. AO16_IOCTL_BUFFER_CLEAR

This service immediately clears the current content from the output buffer. It also clears the associated data overflow and frame overflow status bits. This service does not halt output.

Usage

Argument	Description
request	AO16_IOCTL_BUFFER_CLEAR
arg	Not used.

4.7.4. AO16_IOCTL_BUFFER_MODE

This service configures the board to reuse or discard data after it exits the output buffer.

Usage

Argument	Description
request	AO16_IOCTL_BUFFER_MODE
arg	s32*

Valid argument values are as follows.

Value	Description
-1	Retrieve the current state.
AO16_BUFFER_MODE_CIRC	This refers to circular mode.
AO16_BUFFER_MODE_OPEN	This refers to open mode.

4.7.5. AO16_IOCTL_BUFFER_OVER_DATA

This service operates on the output buffer data overflow status. The board asserts this status if data is written to the output buffer when it is full. The driver is designed to prevent such errors, so they should only appear when such register writes are performed by an application. See section 8.4 (page 54) for additional information.

Usage

Argument	Description
request	AO16_IOCTL_BUFFER_OVER_DATA
arg	s32*

Valid argument values are as follows.

Value	Description
-1	Retrieve the current state.
AO16_BUFFER_OVER_DATA_CHK	Check to see if there was a data overflow.
AO16_BUFFER_OVER_DATA_CLR	Clear a data over flow.

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

Value	Description
AO16_BUFFER_OVER_DATA_NO	A data overflow has not occurred.
AO16_BUFFER_OVER_DATA_YES	A data overflow has occurred.

4.7.6. AO16_IOCTL_BUFFER_OVER_FRAME

This service operates on the output buffer frame overflow status. The board asserts this status if data is written to the output buffer while it is in a closed state. In other words, the output buffer is in circular mode and a write takes place while the buffer is closed. It is an application's responsibility to prevent this from happening. See section 8.4 (page 54) for additional information.

Usage

Argument	Description
request	AO16_IOCTL_BUFFER_OVER_FRAME
arg	s32*

Valid argument values are as follows.

Value	Description
-1	Retrieve the current state.
AO16_BUFFER_OVERFLOW_CHK	Check to see if there was a frame overflow.
AO16_BUFFER_OVERFLOW_CLR	Clear a data over flow.

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

Value	Description
AO16_BUFFER_OVERFLOW_NO	A frame overflow has not occurred.
AO16_BUFFER_OVERFLOW_YES	A frame overflow has occurred.

4.7.7. AO16_IOCTL_BUFFER_SIZE

This service sets the size of the virtual output buffer.

Usage

Argument	Description
request	AO16_IOCTL_BUFFER_SIZE
arg	s32*

Valid argument values are as follows.

Value	Description
-1	Retrieve the current state.
AO16_BUFFER_SIZE_8	This refers to a buffer size of 8 samples deep.
AO16_BUFFER_SIZE_16	This refers to a buffer size of 16 samples deep.
AO16_BUFFER_SIZE_32	This refers to a buffer size of 32 samples deep.
AO16_BUFFER_SIZE_64	This refers to a buffer size of 64 samples deep.
AO16_BUFFER_SIZE_128	This refers to a buffer size of 128 samples deep.
AO16_BUFFER_SIZE_256	This refers to a buffer size of 256 samples deep.
AO16_BUFFER_SIZE_512	This refers to a buffer size of 512 samples deep.
AO16_BUFFER_SIZE_1K	This refers to a buffer size of 1K (1,024) samples deep.
AO16_BUFFER_SIZE_2K	This refers to a buffer size of 2K (2,048) samples deep.
AO16_BUFFER_SIZE_4K	This refers to a buffer size of 4K (4,096) samples deep.
AO16_BUFFER_SIZE_8K	This refers to a buffer size of 8K (8,192) samples deep.
AO16_BUFFER_SIZE_16K	This refers to a buffer size of 16K (16,384) samples deep.
AO16_BUFFER_SIZE_32K	This refers to a buffer size of 32K (32,768) samples deep.
AO16_BUFFER_SIZE_64K	This refers to a buffer size of 64K (65,536) samples deep.
AO16_BUFFER_SIZE_128K	This refers to a buffer size of 128K (131,072) samples deep.
AO16_BUFFER_SIZE_256K	This refers to a buffer size of 256K (262,144) samples deep.

4.7.8. AO16_IOCTL_BUFFER_STATUS

This service retrieves the current output buffer fill level status.

Usage

Argument	Description
request	AO16_IOCTL_BUFFER_STATUS
arg	s32*

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

Value	Description
AO16_BUFFER_STATUS_EMPTY	The output buffer is empty.
AO16_BUFFER_STATUS_1Q_FULL	The output buffer is ¼ full or less.
AO16_BUFFER_STATUS_MEDIUM	The output buffer is between ¼ and ¾ full.
AO16_BUFFER_STATUS_3Q_FULL	The output buffer is ¾ full or more.
AO16_BUFFER_STATUS_FULL	The output buffer is full.

4.7.9. AO16_IOCTL_BURST_ENABLE

This service enables or disables output bursting.

Usage

Argument	Description
request	AO16_IOCTL_BURST_ENABLE
arg	s32*

Valid argument values are as follows.

Value	Description
-1	Retrieve the current state.
AO16_BURST_ENABLE_NO	This refers to output bursting being disabled.
AO16_BURST_ENABLE_YES	This refers to output bursting being enabled.

4.7.10. AO16_IOCTL_BURST_READY

This service reports the output buffer's readiness for an output burst.

Usage

Argument	Description
request	AO16_IOCTL_BURST_READY
arg	s32*

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

Value	Description
AO16_BURST_READY_NO	The output buffer is not ready to output a burst.
AO16_BURST_READY_YES	The output buffer is ready to output a burst.

4.7.11. AO16_IOCTL_BURST_TRIG_SRC

This service configures the source for the bursting trigger.

Usage

Argument	Description
request	AO16_IOCTL_BURST_TRIG_SRC
arg	s32*

Valid argument values are as follows.

Value	Description
-1	Retrieve the current state.
AO16_BURST_TRIG_SRC_EXT	This refers to the external burst trigger input.
AO16_BURST_TRIG_SRC_SW	This refers to the software burst trigger source.

4.7.12. AO16_IOCTL_BURST_TRIGGER

This service initiates a software triggered output burst.

Usage

Argument	Description
request	AO16_IOCTL_BURST_TRIGGER
arg	Not used.

4.7.13. AO16_IOCTL_CBL_ISO_CLOCK_IO

This service configures the operation (the functional isolation) of the cable's Clock I/O signal.

Usage

Argument	Description
request	AO16_IOCTL_CBL_ISO_CLOCK_IO
arg	s32*

Valid argument values are as follows.

Value	Description
-1	Retrieve the current state.
AO16_CBL_ISO_NORM	This refers to the signal being in its default state. †
AO16_CBL_ISO_OUT_0	This refers to the signal being configured to output a logic low.

† If the polarity feature is unsupported, then the *normal* option is always returned.

4.7.14. AO16_IOCTL_CBL_ISO_DAC_CLK_OUT

This service configures the operation (the functional isolation) of the cable's DSC Clock Output signal.

Usage

Argument	Description
request	AO16_IOCTL_CBL_ISO_DAC_CLK_OUT
arg	s32*

Valid argument values are as follows.

Value	Description
-1	Retrieve the current state.
AO16_CBL_ISO_NORM	This refers to the signal being in its default state. †
AO16_CBL_ISO_OUT_0	This refers to the signal being configured to output a logic low.

† If the polarity feature is unsupported, then the *normal* option is always returned.

4.7.15. AO16_IOCTL_CBL_ISO_TRIG_OUT

This service configures the operation (the functional isolation) of the cable's Trigger Output signal.

Usage

Argument	Description
request	AO16_IOCTL_CBL_ISO_TRIG_OUT
arg	s32*

Valid argument values are as follows.

Value	Description
-1	Retrieve the current state.
AO16_CBL_ISO_NORM	This refers to the signal being in its default state. †
AO16_CBL_ISO_OUT_0	This refers to the signal being configured to output a logic low.

† If the polarity feature is unsupported, then the *normal* option is always returned.

4.7.16. AO16_IOCTL_CBL_POL_CLOCK_IO

This service configures the polarity of the cable’s Clock I/O signal.

Usage

Argument	Description
request	AO16_IOCTL_CBL_POL_CLOCK_IO
arg	s32*

Valid argument values are as follows.

Value	Description
-1	Retrieve the current state.
AO16_CBL_POL_INV	This refers to the signal being inverted.
AO16_CBL_POL_NORM	This refers to the signal being in its default state. †

† If the polarity feature is unsupported, then the *normal* option is always returned.

4.7.17. AO16_IOCTL_CBL_POL_DAC_CLK_OUT

This service configures the polarity of the cable’s DAC Clock Out signal.

Usage

Argument	Description
request	AO16_IOCTL_CBL_POL_DAC_CLK_OUT
arg	s32*

Valid argument values are as follows.

Value	Description
-1	Retrieve the current state.
AO16_CBL_POL_INV	This refers to the signal being inverted.
AO16_CBL_POL_NORM	This refers to the signal being in its default state. †

† If the polarity feature is unsupported, then the *normal* option is always returned.

4.7.18. AO16_IOCTL_CBL_POL_TRIG_IN

This service configures the polarity of the cable’s Trigger Input signal.

Usage

Argument	Description
request	AO16_IOCTL_CBL_POL_TRIG_IN
arg	s32*

Valid argument values are as follows.

Value	Passed To API	Returned By API
AO16_CBL_POL_INV		This refers to the signal being inverted.
AO16_CBL_POL_NORM		This refers to the signal being in its default state. †

† If the polarity feature is unsupported, then the *normal* option is always returned.

4.7.19. AO16_IOCTL_CBL_POL_TRIG_OUT

This service configures the polarity of the cable’s Trigger Output signal.

Usage

Argument	Description
request	AO16_IOCTL_CBL_POL_TRIG_OUT
arg	s32*

Valid argument values are as follows.

Value	Passed To API	Returned By API
AO16_CBL_POL_INV		This refers to the signal being inverted.
AO16_CBL_POL_NORM		This refers to the signal being in its default state. †

† If the polarity feature is unsupported, then the *normal* option is always returned.

4.7.20. AO16_IOCTL_CHANNEL_SEL

This service selects the set of channels that generate output.

Usage

Argument	Description
request	AO16_IOCTL_CHANNEL_SEL
arg	s32*

Valid argument values are -1 to retrieve the current enable mask and any combination of bits from 0x0 for no channels to 0xFFFF for all 16 channels. The upper limit is 0xFF for eight channel boards and 0xFFF for 12 channel boards.

4.7.21. AO16_IOCTL_CLOCK_ENABLE

This service enables or disabled output clocking.

Usage

Argument	Description
request	AO16_IOCTL_CLOCK_ENABLE
arg	s32*

Valid argument values are as follows.

Value	Description
-1	Retrieve the current state.
AO16_CLOCK_ENABLE_NO	This refers to output clocking being disabled.
AO16_CLOCK_ENABLE_YES	This refers to output clocking being enabled.

4.7.22. AO16_IOCTL_CLOCK_READY

This service indicates if the board is ready to accept an output clock.

Usage

Argument	Description
request	AO16_IOCTL_CLOCK_READY
arg	s32*

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

Value	Description
AO16_CLOCK_READY_NO	The board is not ready to accept an output clock.
AO16_CLOCK_READY_YES	The board is ready to accept an output clock.

4.7.23. AO16_IOCTL_CLOCK_REF_SRC

This service sets the source for the reference clock.

Usage

Argument	Description
request	AO16_IOCTL_CLOCK_REF_SRC
arg	s32*

Valid argument values are as follows.

Value	Description
-1	Retrieve the current state.
AO16_CLOCK_REF_SRC_ALT	This refers to the alternate source, whose reference is configurable.
AO16_CLOCK_REF_SRC_PRI	This refers to the primary source, whose reference is fixed.

4.7.24. AO16_IOCTL_CLOCK_SRC

This service selects the output clocking source.

Usage

Argument	Description
request	AO16_IOCTL_CLOCK_SRC
arg	s32*

Valid argument values are as follows.

Value	Description
-1	Retrieve the current state.

AO16_CLOCK_SRC_EXT_SW	This refers to the external clock source as well as the software source.
AO16_CLOCK_SRC_INT	This refers to the internal clock source.

4.7.25. AO16_IOCTL_CLOCK_SW

This service generates a single output clock strobe.

Usage

Argument	Description
request	AO16_IOCTL_CLOCK_SW
arg	Not used.

4.7.26. AO16_IOCTL_DATA_FORMAT

This service configures the data encoding format.

Usage

Argument	Description
request	AO16_IOCTL_DATA_FORMAT
arg	s32*

Valid argument values are as follows.

Value	Description
-1	Retrieve the current setting.
AO16_DATA_FORMAT_2S_COMP	This refers to the Twos Compliment data format.
AO16_DATA_FORMAT_OFF_BIN	This refers to the Offset Binary encoding format.

4.7.27. AO16_IOCTL_GROUND_SENSE

This service enables or disables use of remote ground sense.

Usage

Argument	Description
request	AO16_IOCTL_GROUND_SENSE
arg	s32*

Valid argument values are as follows.

Value	Description
-1	Retrieve the current state.
AO16_GROUND_SENSE_DISABLE	This refers to remote ground sense being disabled.
AO16_GROUND_SENSE_REMOTE	This refers to remote ground sense being enabled.

4.7.28. AO16_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.

NOTE: For boards with the high voltage range feature (the -HL ordering option) the initialization service sets the voltage range option to the ± 5 -volt range before returning. For low voltage range boards, the voltage range after initialization is the ± 2.5 -volt option.

Usage

Argument	Description
request	AO16_IOCTL_INITIALIZE
arg	Not used.

4.7.29. AO16_IOCTL_IRQ_SEL

This service configures the firmware's interrupt source selection.

Usage

Argument	Description
request	AO16_IOCTL_IRQ_SEL
arg	s32*

Valid argument values are as follows.

Value	Description
-1	Retrieve the current setting.
AO16_IRQ_AUTOCAL_DONE	This refers to the completion of an autocalibration cycle.
AO16_IRQ_BUF_1Q_FULL	This refers to the buffer falling to the $\frac{1}{4}$ full level.
AO16_IRQ_BUF_3Q_FULL	This refers to the buffer rising to the $\frac{3}{4}$ full level.
AO16_IRQ_BUF_EMPTY	This refers to the buffer becoming empty.
AO16_IRQ_BURST_TRIG_READY	This refers to the readiness of the board to accept a burst trigger.
AO16_IRQ_INIT_DONE	This refers to the completion of an initialization cycle.
AO16_IRQ_LOAD_READY	This refers to the condition where the output buffer becomes ready to accept data.
AO16_IRQ_LOAD_READY_END	This refers to the condition where the output buffer is no longer ready to accept data.

4.7.30. AO16_IOCTL_LOAD_READY

This service reports if the output buffer is ready to receive data.

Usage

Argument	Description
request	AO16_IOCTL_LOAD_READY
arg	s32*

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

Value	Description
AO16_LOAD_READY_NO	The buffer is not ready to receive data.
AO16_LOAD_READY_YES	The buffer is ready to receive data.

4.7.31. AO16_IOCTL_LOAD_REQUEST

This service requests access to the output buffer, which should be configured for circular operation. The driver requests access and returns immediately rather than waiting for access to be granted.

Usage

Argument	Description
request	AO16_IOCTL_LOAD_REQUEST
arg	Not used.

4.7.32. AO16_IOCTL_NCLK

This service configures the alternate reference frequency by setting the NCLK divider value.

Usage

Argument	Description
request	AO16_IOCTL_NCLK
arg	s32*

Valid argument values are -1 to retrieve the current setting and 0x0 to 0x1FF.

4.7.33. AO16_IOCTL_NRATE

This service configures the rate generator by setting the NRATE divider value.

Usage

Argument	Description
request	AO16_IOCTL_NRATE
arg	s32*

Valid argument values are -1, to retrieve the current setting, and some minimum up through 0x3FFFF. The minimum appropriate value is based on the board's reference frequency. See the table below. If the board has a custom frequency, the driver can't know for certain what that frequency is and thus, may report an incorrect minimum NRATE value. Refer to the AO16_IOCTL_QUERY_IOCTL service (section 4.7.36, page 36) for additional information.

NOTE: If the driver reports an incorrect reference frequency, then an application may have to apply the desired NRATE value manually by direct access to the Sample Rate Register (SRR).

Board Model	Reference Frequency	NRATE Minimum	Note
16AO16	45,000,000 Hz	100	Default
	49,152,000 Hz	110	Default custom value
	44,982,000 Hz	100	Custom value

4.7.34. AO16_IOCTL_OUTPUT_FILTER

This service selects the output filter option.

Usage

Argument	Description
request	AO16_IOCTL_OUTPUT_FILTER
arg	s32*

Valid argument values are as follows.

Value	Passed To API	Returned By API
-1	Retrieve the current state.	The feature is unsupported.
AO16_OUTPUT_FILTER_A	This refers to sequential output.	
AO16_OUTPUT_FILTER_B	This refers to simultaneous output.	
AO16_OUTPUT_FILTER_NONE	This refers to simultaneous output.	

4.7.35. AO16_IOCTL_OUTPUT_MODE

This service selects the mode for the output clock.

Usage

Argument	Description
request	AO16_IOCTL_OUTPUT_MODE
arg	s32*

Valid argument values are as follows.

Value	Description
-1	Retrieve the current state.
AO16_OUTPUT_MODE_SEQ	This refers to sequential output.
AO16_OUTPUT_MODE_SIM	This refers to simultaneous output.

4.7.36. AO16_IOCTL_QUERY

This service queries the driver for various pieces of information about the board and the driver.

Usage

Argument	Description
request	AO16_IOCTL_QUERY
arg	s32*

Valid argument values are as follows.

Value	Description
AO16_QUERY_AUTOCAL_MS	This returns the maximum duration of the Autocalibration cycle in milliseconds.
AO16_QUERY_CABLE_INVERT_4	This returns an indication if the firmware supports inverting the four cable interface signals Trigger Input, Trigger Output, DAC Clock Out and Clock I/O.
AO16_QUERY_CABLE_PASSIVE_3	This returns an indication if the firmware supports isolating the normal interface functionality of the three cable interface signals Trigger Output, DAC Clock Out and Clock I/O.
AO16_QUERY_CHANNEL_MASK	This returns the mask of valid channel enable bits.

AO16_QUERY_CHANNEL_MAX	This returns the maximum number of output channels supported by the board, which may be more than the board's current configuration.
AO16_QUERY_CHANNEL_QTY	This returns the actual number of output channels on the current board. If the value returned is -1, then the driver was unable to determine the number of channels.
AO16_QUERY_COUNT	This returns the number of query options supported by the IOCTL service.
AO16_QUERY_DEVICE_TYPE	This returns the identifier value for the board's type. This should be GSC_DEV_TYPE_16AO16.
AO16_QUERY_DIFFERENTIAL	This returns a value indicating if the board has differential outputs (0 = no, 1 = yes).
AO16_QUERY_DMDMA	This returns a value indicating if Demand Mode DMA is supported (0 = no, 1 = yes).
AO16_QUERY_FIFO_SIZE	This returns the size of the output buffer in 32-bit A/D values.
AO16_QUERY_FILTER	This returns an indicator of the board's output filter options. Valid return values are listed in a table below.
AO16_QUERY_FREQ_DEFAULT	This returns the default reference frequency. †
AO16_QUERY_FSAMP_MAX	This gives the maximum FSAMP value in S/S.
AO16_QUERY_FSAMP_MIN	This gives the minimum FSAMP value in S/S. ‡
AO16_QUERY_INIT_MS	This returns the duration of a board initialization in milliseconds.
AO16_QUERY_LAST	This gives the last enumeration value.
AO16_QUERY_MODEL	This returns the device's model type. See below.
AO16_QUERY_NCLK_MASK	This returns the mask for the alternate reference source's NCLK divider value.
AO16_QUERY_NCLK_MAX	This returns the maximum valid NCLK value.
AO16_QUERY_NCLK_MIN	This returns the minimum valid NCLK value.
AO16_QUERY_NRATE_MASK	This returns the mask for the board's NRATE divider value.
AO16_QUERY_NRATE_MAX	This returns the maximum supported NRATE divider value.
AO16_QUERY_NRATE_MIN	This returns the minimum supported NRATE divider value. ‡
AO16_QUERY_OUTPUT_CAPACITY	This indicates the device's support for high voltage vs high current capacity capability. See below.
AO16_QUERY_OUTPUT_FILTER	This indicates if the device supports selecting the Output Filter.
AO16_QUERY_VOLT_RANGE	This returns the device's supported voltage range. See below.
AO16_QUERY_WATCHDOG	This indicates if the device supports the Watchdog Bit feature.

† The 16AO16 supports more than one custom frequency. The board itself does not reveal which custom frequency is in use. It only reveals that the reference frequency is the default of a custom value.

‡ This value is affected by the default reference frequency, which may not be known by the driver. †

NOTE: If the 16AO16 in use has a custom reference frequency other than that selected by the driver, then this can be corrected by adjusting the code in the driver's `device.c` source file.

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

Value	Description
AO16_IOCTL_QUERY_ERROR	Either there was a processing error or the query option is unrecognized.

Valid return values for the filter query are as follows.

Value	Description
AO16_FILTER_NONE	No custom filter option is installed.
AO16_FILTER_10KHZ	A 10 KHz filter is installed.

AO16_FILTER_100KHZ	A 100 KHz filter is installed.
AO16_FILTER_F1	The device has the F1 filters documented in the reference manual.
AO16_FILTER_F2	The device has the F2 filters documented in the reference manual.
AO16_FILTER_F3	The device has the F3 filters documented in the reference manual.
AO16_FILTER_F4	The device has the F4 filters documented in the reference manual.

Valid return values for the model query are as follows.

Value	Description
AO16_MODEL_16AO16	This indicates that the device is a 16AO16, though the form factor is unknown.
AO16_MODEL_16AO16FLV	This indicates that the device is a 16AO16FLV, though the form factor is unknown.

Valid return values for the output capacity query are as follows.

Value	Description
-1	This feature is not supported.
AO16_OUTPUT_CAPACITY_HI_CURRENT	The board is factory configured for the high current option, which supports voltage settings of $\pm 1.5V$ and $\pm 2.5V$.
AO16_OUTPUT_CAPACITY_HI_LEVEL	The board is factory configured for the high-level option, which supports voltage settings of $\pm 5V$ and $\pm 10V$.

Valid return values for the voltage range query are as follows.

Value	Description
-1	This feature is not supported.
AO16_VOLT_RANGE_LOW	This refers to the low voltage range, which includes selection options of $\pm 1.25V$, $\pm 2.5V$, $\pm 5\pm$ and $\pm 10V$.
AO16_VOLT_RANGE_HIGH	This refers to the high voltage range, which includes selection options of $\pm 5V$, $\pm 10\pm$ and $\pm 20V$.

4.7.37. AO16_IOCTL_RANGE

This service sets the output voltage range.

NOTE: For boards with the high voltage range feature (the -HL ordering option) the initialization service sets the voltage range option to the ± 5 -volt range before returning. For low voltage range boards, the voltage range after initialization is the ± 2.5 -volt option.

Usage

Argument	Description
request	AO16_IOCTL_RANGE
arg	s32*

Valid argument values are as follows.

Value	Description
-1	Retrieve the current state.
AO16_RANGE_1_25	This refers to the voltage range of ± 1.25 volts.
AO16_RANGE_2_5	This refers to the voltage range of ± 2.5 volts.

AO16_RANGE_5	This refers to the voltage range of ± 5 volts.
AO16_RANGE_10	This refers to the voltage range of ± 10 volts.

4.7.38. AO16_IOCTL_REG_MOD

This service performs a read-modify-write of a 16AO16 register. This includes only the GSC firmware registers. The PCI and PLX Feature Set Registers are read-only. Refer to `16ao16.h` for the complete list of GSC firmware registers.

Usage

Argument	Description
request	AO16_IOCTL_REG_MOD
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 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 bit is modified. If a bit here is zero, then the respective register bit is unmodified.

4.7.39. AO16_IOCTL_REG_READ

This service reads the value of a 16AO16 register. This includes the PCI registers, the PLX Feature Set Registers and the GSC firmware registers. Refer to `16ao16.h` and `gsc_pci9056.h` for the complete list of accessible registers.

Usage

Argument	Description
request	AO16_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.

4.7.40. AO16_IOCTL_REG_WRITE

This service writes a value to a 16AO16 register. This includes only the GSC firmware registers. The PCI and PLX Feature Set Registers are read-only. Refer to `16ao16.h` for a complete list of the GSC firmware registers.

Usage

Argument	Description
request	AO16_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.

4.7.41. AO16_IOCTL_TX_IO_ABORT

This service aborts an ongoing `write()` request.

Usage

Argument	Description
request	AO16_IOCTL_TX_IO_ABORT
arg	<code>s32*</code>

The results are reported as one of the following values.

Value	Description
<code>AO16_IO_ABORT_NO</code>	A <code>write()</code> request was not aborted as none were ongoing.
<code>AO16_IO_ABORT_YES</code>	An ongoing <code>write()</code> request was aborted.

4.7.42. AO16_IOCTL_TX_IO_MODE

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

Usage

Argument	Description
request	AO16_IOCTL_TX_IO_MODE
arg	<code>s32*</code>

Valid argument values are as follows.

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

† Demand Mode DMA is not supported on all boards. Use the DMDMA query option to find out if Demand Mode DMA is supported (section 4.7.36, page 36).

4.7.43. AO16_IOCTL_TX_IO_OVER_DATA

This service configures the write service to check for an output buffer data overflow before performing write operations. Sample data is lost when there is a data overflow. See section 8.4 (page 54) for additional information.

NOTE: The check for a data overflow is performed upon entry to the write service. The write service does not check for data overflows that occur while the write is in progress. For in-progress data overflows an application must perform the check manually or wait for the check performed by a subsequent write request.

Usage

Argument	Description
request	AO16_IOCTL_TX_IO_OVER_DATA
arg	s32*

Valid argument values are as follows.

Value	Description
-1	Retrieve the current setting.
AO16_TX_IO_OVER_DATA_CHECK	This refers to the check being performed. This is the default.
AO16_TX_IO_OVER_DATA_IGNORE	This refers to the check not being performed.

4.7.44. AO16_IOCTL_TX_IO_OVER_FRAME

This service configures the write service to check for an output buffer frame overflow before performing write operations. Sample data is lost when there is a frame overflow. See section 8.4 (page 54) for additional information.

NOTE: The check for a frame overflow is performed upon entry to the write service. The write service does not check for frame overflows that occur while the write is in progress. For in-progress frame overflows an application must perform the check manually or wait for the check performed by a subsequent write request.

Usage

Argument	Description
request	AO16_IOCTL_TX_IO_OVER_FRAME
arg	s32*

Valid argument values are as follows.

Value	Description
-1	Retrieve the current setting.

AO16_TX_IO_OVER_FRAME_CHECK	This refers to the check being performed. This is the default.
AO16_TX_IO_OVER_FRAME_IGNORE	This refers to the check not being performed.

4.7.45. AO16_IOCTL_TX_IO_TIMEOUT

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

Usage

Argument	Description
request	AO16_IOCTL_TX_IO_TIMEOUT
arg	s32*

Valid argument values are in the range from zero to 3600, -1, and AO16_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 AO16_IOCTL_TIMEOUT_INFINITE is used, then the driver will wait indefinitely rather than timing out. The default is 10 seconds.

4.7.46. AO16_IOCTL_WAIT_CANCEL

This service resumes all threads blocked via AO16_IOCTL_WAIT_EVENT IOCTL calls (section 4.7.47, page 43), 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	AO16_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 4.7.47.2 on page 44.
gsc	This specifies the set of AO16_WAIT_GSC_* events whose wait requests are to be cancelled. Refer to section 4.7.47.3 on page 44.
alt	This is unused by the 16AO16 driver and should be zero.

io	This specifies the set of AO16_WAIT_IO_* events whose wait requests are to be cancelled. Refer to section 4.7.47.4 on page 44.
timeout_ms	This is unused by wait cancel operations.
count	Upon return this indicates the number of waits that were cancelled.

4.7.47. AO16_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
request	AO16_IOCTL_WAIT_EVENT
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 must initially be zero. Upon return this indicates the reason that the thread was resumed. Refer to section 4.7.47.1 on page 44.
main	This specifies any number of GSC_WAIT_MAIN_* events that the thread is to wait for. Refer to section 4.7.47.2 on page 44.
gsc	This specifies any number of AO16_WAIT_GSC_* events that the thread is to wait for. Refer to section 4.7.47.3 on page 44.
alt	This is unused by the 16AO16 driver and must be zero.
io	This specifies any number of AO16_WAIT_IO_* events that the thread is to wait for. Refer to section 4.7.47.4 on page 44.
timeout_ms	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.
count	This is unused by wait event operations and must be zero.

4.7.47.1. `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.

4.7.47.2. `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 16AO16 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 16AO16.
<code>GSC_WAIT_MAIN_PCI</code>	This refers to any interrupt generated by the 16AO16.
<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.

4.7.47.3. `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 Board Control Register. Applications are responsible for selecting the desired interrupt options. Refer to `AO16_IOCTL_IRQ_SEL` (section 4.7.29, page 34).

Value	Description
<code>AO16_WAIT_GSC_AUTOCAL_DONE</code>	This refers to the completion of an autocalibration cycle.
<code>AO16_WAIT_GSC_BUF_1Q_FULL</code>	This refers to the buffer falling to the 1/4 full level.
<code>AO16_WAIT_GSC_BUF_3Q_FULL</code>	This refers to the buffer rising to the 3/4 full level.
<code>AO16_WAIT_GSC_BUF_EMPTY</code>	This refers to the buffer becoming empty.
<code>AO16_WAIT_GSC_BURST_TRIG_READY</code>	This refers to the readiness of the board to accept a burst trigger.
<code>AO16_WAIT_GSC_INIT_DONE</code>	This refers to the completion of an initialization cycle.
<code>AO16_WAIT_GSC_LOAD_READY</code>	This refers to the condition where the output buffer becomes ready to accept data.
<code>AO16_WAIT_GSC_LOAD_READY_END</code>	This refers to the condition where the output buffer is no longer ready to accept data.

4.7.47.4. `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
<code>AO16_WAIT_IO_TX_ABORT</code>	This refers to write requests which have been aborted.
<code>AO16_WAIT_IO_TX_DONE</code>	This refers to write requests which have been satisfied.
<code>AO16_WAIT_IO_TX_ERROR</code>	This refers to write requests which end due to an error.
<code>AO16_WAIT_IO_TX_TIMEOUT</code>	This refers to write requests which end due to the timeout period lapse.

4.7.48. AO16_IOCTL_WAIT_STATUS

This service counts all threads blocked via the AO16_IOCTL_WAIT_EVENT IOCTL service (section 4.7.47, page 43), 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	AO16_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 4.7.47.2 on page 44.
gsc	This specifies the set of AO16_WAIT_GSC_* events whose wait requests are to be counted. Refer to section 4.7.47.3 on page 44.
alt	This is unused by the 16AO16 driver and should be zero.
io	This specifies the set of AO16_WAIT_IO_* events whose wait requests are to be counted. Refer to section 4.7.47.4 on page 44.
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.

4.7.49. AO16_IOCTL_WATCHDOG_ENABLE

This service enables or disables the Watchdog Bit feature, when supported by firmware.

Usage

Argument	Description
request	AO16_IOCTL_WATCHDOG_ENABLE
arg	s32*

Valid argument values are as follows.

Value	Passed To API	Returned By API
-1	Retrieve the current setting.	The feature is unsupported.
AO16_WATCHDOG_ENABLE_NO	This disables the Watchdog Bit feature.	
AO16_WATCHDOG_ENABLE_YES	This enables the Watchdog Bit feature.	

4.7.50. AO16_IOCTL_WATCHDOG_OUTPUT

This service set the Watchdog Bit output level, when supported by firmware.

Usage

Argument	Description
request	AO16_IOCTL_WATCHDOG_OUTPUT
arg	s32*

Valid argument values are as follows.

Value	Passed To API	Returned By API
-1	Retrieve the current setting.	The feature is unsupported.
AO16_WATCHDOG_OUTPUT_0	This sets the Watchdog Bit output level to low.	
AO16_WATCHDOG_OUTPUT_1	This sets the Watchdog Bit output level to high.	

4.7.51. AO16_IOCTL_XCVR_TYPE

This service selects the transceiver type used by the clock and trigger lines on the cable interface.

Usage

Argument	Description
request	AO16_IOCTL_XCVR_TYPE
arg	s32*

Valid argument values are as follows.

Value	Description
-1	Retrieve the current setting.
AO16_XCVR_TYPE_LVDS	This refers to LVDS transceivers.
AO16_XCVR_TYPE_TTL	This refers to TTL transceivers.

5. The Driver

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

5.1. Files

The device driver files are summarized in the table below.

Description	Files	Location
Source Files	*.c, *.h...	.../driver/
Header File	16ao16.h	
Driver File	16ao16.ko † 16ao16.o ‡	

† This is for kernel versions 2.6 and later.

‡ This is for kernel versions 2.4 and earlier.

5.2. Build

NOTE: Building the driver requires installation of the kernel headers and possibly other packages.

The device driver is built via the Overall Make Script (section 2.7, page 13), but can be built separately following the below steps.

1. Change to the directory where the driver and its sources are installed (.../driver/).
2. Remove existing build targets by issuing the below command.

```
make clean
```

3. Build the driver by issuing the below command.

```
make
```

NOTE: Due to the differences between the many Linux distributions some build errors may occur. These errors may include system header location differences, which should be easily corrected.

5.3. Startup

NOTE: The driver will have to be built before being used as it is provided in source form only.

The startup script used in this procedure is designed to load the device driver and create fresh device nodes. This is accomplished by unloading the current driver, if loaded, and then loading the accompanying driver executable. In addition, the script deletes and recreates the device nodes. This is done to ensure that the device nodes in use have the same major number as assigned dynamically to the driver by the kernel, and so that the number of device nodes corresponds to the number of boards identified by the driver.

5.3.1. Manual Driver Startup Procedures

Start the driver manually by following the below listed steps.

NOTE: The following steps may require elevated privileges.

1. Change to the directory where the driver sources are installed (.../driver/).
2. Install the driver module and create the device nodes by executing the below command. If any errors are encountered then an appropriate error message will be displayed.

```
./start
```

NOTE: This script must be executed each time the host is booted.

NOTE: The 16AO16 device node major number is assigned dynamically by the kernel. The minor numbers and the device node suffix numbers are index numbers beginning with zero, and increase by one for each additional board installed.

3. Verify that the device driver module has been loaded by issuing the below command and examining the output. The module name `16ao16` should be included in the output.

```
lsmod
```

4. Verify that the device nodes have been created by issuing the below command and examining the output. The output should include one node for each installed board.

```
ls -l /dev/16ao16.*
```

5.3.2. Automatic Driver Startup Procedures

Start the driver automatically with each system reboot by following the below listed steps.

1. Locate and edit the system startup script `rc.local`, which should be in the `/etc/rc.d/` directory. Modify the file by adding the below line so that it is executed with every reboot. The example is based on the driver being installed in `/usr/src/linux/drivers/`, though it may have been installed elsewhere.

```
/usr/src/linux/drivers/16ao16/driver/start
```

NOTE: For `systemd` installations the file `rc.local` may be located under the `/etc/` directory rather than under `/etc/rc.d/`.

2. Load the driver and create the required device nodes by rebooting the system.
3. Verify that the driver is loaded and that the device nodes have been created. Do this by following the verification steps given in the manual startup procedures.

5.3.2.1. File `rc.local` Not Present

Some distributions may not install a default version of `rc.local`. Some may not even create the directory `/etc/rc.d/`. If the directory is not present, then it may be created. The directory must be created with the owner and group set to `root`. The directory permissions must be set to `rxr-xr-x`. If the file `/etc/rc.d/rc.local` is not present, then it too may be created. The file must also be created with the owner and group set to `root`. Additionally, the file permissions must also be set to `rxr-xr-x`. After the directory and file are created as described, reboot to verify boot time loading of the driver. Here is an example of a default version of `rc.local`.

```
#!/bin/bash
# Add your local content here.
```

5.3.2.2. Default `rc.local` File Permissions

The `rc.local` script may fail to run at boot time because some distributions install a default version of the file without execute permissions. Without execute permissions, boot time invocation of the script fails, which inhibits boot time loading of the driver. If this is the case, then change the file permissions to `rwxr-xr-x`. After the file permissions are adjusted as described, reboot to verify boot time loading of the driver.

5.3.2.3. `systemd` Installations

With the advent of the `systemd` startup implementation, `rc.local` may be accessed via a `systemd` startup service. The service name may be `rc-local`, `rc-local.service` or something similar. This service may or may not be enabled by default. If the service is disabled, then the script will not execute, which prevents boot time loading of the driver. The service can be enabled with the below command line. After the service is enabled, reboot to verify boot time loading of the driver.

```
systemctl enable rc-local
```

NOTE: For `systemd` installations the file `rc.local` may be located under the `/etc/` directory rather than under `/etc/rc.d/`.

5.3.2.4. `systemd` and `rc.local` Timing

If the above steps have been performed but the driver still does not start then examine the `dmesg` output for driver messages. If the output shows that the driver starts and immediately stops, then the problem may be timing. That is, since `systemd` doesn't serialize startup initialization as done in the past, driver loading may fail if required services have not completed their own initialization. If this is the problem, then it may be corrected simply by inserting a delay in `rc.local` prior to it calling the driver's start script (i.e., `sleep` for one or more seconds).

5.3.2.5. SELinux Implications

If not disabled, then SELinux may prevent boot time loading of the driver. If this is the case, then it can be verified and corrected using SELinux related tools and utilities. First, install the necessary software using the below command. (As necessary, replace the `yum` command line with that which is available for your distribution.)

```
yum install setroubleshoot setools
```

Next, run the below command to determine if SELinux is preventing the driver from loading at boot time.

```
sealert -a /var/log/audit/audit.log
```

If SELinux is preventing the driver from loading, then the output from the above command should include a reference to the driver's start script, the `insmod` command that loads the driver or the name of the driver executable. If so, then the output should also indicate the commands necessary to resolve the issue. The following is an example of the instructions given when the culprit is `insmod`, which is the start script command that loads the driver. After running these commands reboot the system to verify boot time loading of the driver.

```
ausearch -c 'insmod' --raw | audit2allow -M my-insmod
semodule -X 300 -i my-insmod.pp
```

5.4. Verification

Follow the below steps to verify that the driver has been properly installed and started.

1. Verify that the file `/proc/16ao16` is present. If the file is present then the driver is loaded and running. Verify the file's presence by viewing its content with the below command.

```
cat /proc/16ao16
```

5.5. Version

The driver version number can be obtained in a variety of ways. It is reported by the driver both when the driver is loaded and when it is unloaded (depending on kernel configuration options, this may be visible only in places such as `/var/log/messages`). It is reported in the text file `/proc/16ao16` while the driver is loaded and running. The version number is also given in the file `release.txt` in the root install directory.

5.6. Shutdown

Shutdown the driver following the below listed steps.

NOTE: The following steps may require elevated privileges.

1. If the driver is currently loaded then issue the below command to unload the driver.

```
rmmod 16ao16
```

2. Verify that the driver module has been unloaded by issuing the below command. The module name `16ao16` should not be in the listed output.

```
lsmod
```

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
Source Files	*.c, *.h/docsrc/
Header File	16ao16_dsl.h	.../include/
Library File	16ao16_dsl.a	.../lib/

6.2. Build

The library is built via the Overall Make Script (section 2.7, page 13), but can be built separately following the below steps.

1. Change to the directory where the documentation sources are installed (.../docsrc/).
2. Remove existing build targets by issuing the below command.

```
make clean
```

3. Compile the sample files and build the library by issuing the below command.

```
make
```

4. Rebuild the Main Library (section 3.2.1, page 16).

6.3. Library Use

The library is used both at application compile time and at application link time. At compile time include the above listed header file in each source file using a component of the library interface. At link time include the above listed static library file with the objects being linked with the application.

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 `ao16_open()` there is the utility file `open.c` containing the utility function `ao16_open_util()`. The naming pattern is as follows: API function `ao16_xxxx()`, utility file name `xxxx.c`, utility function `ao16_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 `AO16_IOCTL_QUERY` there is the utility file `query.c` containing the utility function `ao16_query()`. The naming pattern is as follows: IOCTL code `AO16_IOCTL_XXXX`, utility file name `xxxx.c`, utility function `ao16_xxxx()`.

7.1. Files

The utility files are summarized in the table below.

Description	Files	Location
Source Files	*.c, *.h/utils/
Header File	16ao16_utils.h	.../include/
Library Files	16ao16_utils.a gsc_utils.a os_utils.a plx_utils.a	.../lib/

7.2. Build

The libraries are built via the Overall Make Script (section 2.7, page 13), but can be built separately following the below steps.

1. Change to the directory where the utility sources are installed (.../utils/).
2. Remove existing build targets by issuing the below command.

```
make clean
```

3. Compile the sample files and build the library by issuing the below command.

```
make
```

4. Rebuild the Main Library (section 3.2.1, page 16).

7.3. Library Use

The library is used both at application compile time and at application link time. At compile time include the above listed header file in each source file using a component of the library interface. At link time include the above listed static library file with the objects being linked with the application.

8. Operating Information

This section explains some basic operational procedures for using the 16AO16. This is in no way intended to be a comprehensive guide. This is simply to address a very few issues relating to their use.

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
Application	id	.../id/

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
Function	ao16_reg_list()	Source File
Source File	reg.c	.../utils/
Header File	16ao16_utils.h	.../include/
Library File	16ao16_utils.a	.../lib/

8.2. 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.

Item	Name/File	Location
Function	ao16_config_ao()	Source File
Source File	config_ao.c	.../utils/
Header File	16ao16_utils.h	.../include/
Library File	16ao16_utils.a	.../lib/

8.3. Data Transfer Modes

All device I/O requests move data through intermediate driver buffers on its way between the device and application memory. The data is processed in chunks no larger than the size of this intermediate buffer. The process used to

perform this transfer is according to the I/O mode selection. Movement of data between the application buffers and the intermediate driver buffers is performed by the kernel.

8.3.1. PIO - Programmed I/O

In this mode data is transferred using repetitive register accesses. This is most applicable for low throughput requirements or for small transfer requests. The driver continues the operation until either the I/O request is fulfilled or the I/O timeout expires, whichever occurs first. This is generally the least efficient mode, but for very small transfers it is more efficient than DMA.

8.3.2. BMDMA - Block Mode DMA

For DMA transfers, hardware onboard the 16AO16 is used to transfer the data without processor intervention. In this mode the driver checks for available space in the output buffer. Depending on the size of the write request, the driver may break the request into smaller transfers in order to ensure data integrity. When sufficient space is available a DMA transfer is performed. The volume of data moved in a single request is based upon the amount of data remaining in the request and the amount of space available in the buffer. If the remaining request will fit within the available space, then the data is transferred. Otherwise, the volume of data that is transferred is based on the buffer fill level. If the buffer is full, then driver waits one system timer tick before trying again. The process is repeated until the data is exhausted or the I/O timeout expires, whichever occurs first.

8.3.3. DMDMA - Demand Mode DMA

In Demand Mode DMA, data is moved from the intermediate buffer to the output buffer in a single DMA transfer that occurs over time as the data appears in the output buffer. The process is repeated until the data is exhausted or the I/O timeout expires, whichever occurs first.

8.4. Output Overflow Errors

When output overflow errors occur, they are discovered either because the write service (section 4.6.6, page 24) returns an error status or because a status query (section 4.7.43, page 41 and section 4.7.44, page 41) is made. The write service error status returned is `-5`. This is equivalent to `-EIO`, where the `EIO` is from `errno.h`. While these errors always result in this error status, this error status can also come from other errors within the driver or the kernel. When this value is returned, an overflow is the most likely cause. Overflow queries may be made at any time, though there may seldom be a need. Both error conditions result in data loss, the result of which may likely be desynchronization of the output data with the desired output channels.

8.4.1. Data Overflow

The driver is designed so that write requests should never overflow the output buffer. If a data overflow does occur, it is because the application explicitly wrote to the Analog Output Data Buffer Register while a write request was active. When this occurs, data is lost. The volume of data lost corresponds to the number of values written manually to the output buffer register. The position, or positions, of loss in the data stream is essentially unknown.

8.4.2. Frame Overflow

Frame overflows can occur only when operating the output buffer in recirculating mode. When so doing, applications must ensure both that an open request is granted before the write begins and that the write concludes before the buffer closes again. Data is lost when it is written while the output buffer is closed. When this occurs, the volume of data lost is unknown. Here too, the position, or positions, of loss in the data stream is essentially unknown.

8.4.3. Recovery Action

It is common that an application take action to recover from output data loss. The minimum that must be done is clearing the status bits. This can be done by clearing the buffer (section 4.7.3, page 25), which clears both status bits, or by clearing each status bit individually (section 4.7.43, page 41 and section 4.7.44, page 41). Clearing the buffer may be the preferred response as applications may need to reset active output channels to known, safe voltage levels. Afterwards applications can restart the output stream to resume normal operation.

9. Sample Applications

The driver archive includes a variety of sample and test applications. While they are provided without support and without any external documentation, any problems reported will be addressed as time permits. The applications are command line based and produce text output for display on a console. All of the applications are built via the Overall Make Script (section 2.7, page 13), but each may be built individually by changing to its respective directory and issuing the commands “make clean” and “make all”. The initial output from each application includes information on its supported command line arguments. The following gives a brief overview of each application.

9.1. aout - Analog Output - .../aout/

This application outputs a repeating pattern on the first four output channels. The pattern is different for each channel, though they are synchronized at the same modest rate.

9.2. clockout - Clock Output - .../clockout/

This application configures the board to drive the digital output signals for a user specified period of time. This is done to facilitate setup of test equipment to capture those signals during actual use.

9.3. fsamp - Sample Rate - .../fsamp/

This application reports the device configuration required to produce a user specified sample rate.

9.4. id - Identify Board - .../id/

This application reports detailed board identification information. This can be used with tech support to help identify as much technical information about the board as possible from software.

9.5. mcao - Multi-Channel Analog Output - .../mcao/

This application configures a specified number of channels for operation, and then outputs the designated wave patterns on the designated channels.

9.6. regs - Register Access - .../regs/

This application provides menu based interactive access to the board’s registers, and reports other pertinent information to the console.

9.7. sbtest - Single Board Test - .../sbtest/

This application performs functional testing of the driver and a user specified board, at least to the extent possible with just a single board and no additional equipment.

9.8. txrate - Transmit Rate - .../txrate/

This application configures the board for its highest output sample rate then writes output as fast as possible. The purpose is to measure the peak sustainable output rate for the host, per the provided command line arguments.

Document History

Revision	Description
February 6, 2025	Updated to version 3.12.111.50.0. Minor editorial changes. Updated the kernel support table. Updated the information on the NRATE setting. Updated the information on the Query IOCTL service. Updated information on overflow errors and recovery. Removed the “util_” prefix from the utility source file names. Added information on data and frame overflow errors and how to handle them.
August 1, 2024	Updated to version 3.11.111.50.0. Numerous, minor editorial changes. Updated the kernel support table. Added information on the shared object script (section 3.2.3, page 17).
June 28, 2023	Updated to version 3.10.104.47.0. Updated the kernel support table. Numerous, minor editorial changes. Updated the description of the Output Buffer Clear service. Updated the description of the Autocalibration service. Renamed all Auto_Cal content to Autocal. Renamed all Auto_Cal_Sts content to Autocal_Status.
January 30, 2023	Updated to version 3.9.102.44.0. Added notes regarding the autocalibration IOCTL service.
January 23, 2023	Updated to version 3.8.102.44.0. Updated the information for the open and close calls. Reorganized the operations section and added I/O transfer information. Minor editorial alterations. Added support for the 16AO16FLV. Added support for various features and query options: Voltage Range, Cable Invert, Cable Passive, Model, Output Capacity, Output Filter and Watchdog Bit.
July 6, 2022	Updated to version 3.7.100.42.0. Expanded automatic startup information. Minor editorial alterations. Updated information on the main library and its use. Updated the kernel support table. Added section on environment variables.
February 24, 2021	Updated to version 3.6.93.35.0. Updated the kernel support table. Minor editorial changes. Added WAIT_EVENT note. Corrected the description of the XCVR_TYPE service. Expanded automatic startup information. Added notes on the voltage range selection following initialization.
July 15, 2019	Updated to version 3.5.86.28.0. Updated the kernel support table. Minor editorial updates. Added a licensing subsection.
May 16, 2019	Updated to version 3.4.85.27.0. Minor editorial changes. Minor paragraph updates. Updated the software architecture figure.
October 24, 2018	Updated to version 3.3.81.26.1. Various editorial changes. Added debugging aids. Changed GSC WAIT IO XXX to AO16 WAIT IO XXX.
October 23, 2018	Updated to version 3.3.81.26.0. Updated the inside cover page. Updated Block Mode DMA macro and associated information.
June 7, 2018	Updated to version 3.2.77.22.1. The API Library is now implemented as a shared library.
June 1, 2018	Updated to version 3.2.77.22.0. Document reorganization. Numerous, minor editorial changes. Updated the CPU and kernel support section.
November 30, 2016	Updated to version 3.1.68.18.0. Removed the built field from the /proc/ file. Updated the kernel support table. Updated the command line arguments for the fsamp, aout and txrate sample applications. Organized the sample applications alphabetically. Added the mcao sample application. Updated the usage of the Wait Event timeout_ms field. Updated material on the open call. Added open access mode descriptions. Added support for infinite I/O timeouts. Added a section for general operating information. Made various miscellaneous updates. Some document reorganization.
September 14, 2015	Updated to version 3.0.60.8.0. Removed double underscore that prefaced various data types.
October 23, 2014	Updated to version 2.6.57.0.
July 18, 2014	Updated to version 2.6.53.0. Changed the device name from 16ao16n to 16ao16.n. Added support for Demand Mode DMA. Expanded the Voltage Range IOCTL service. Added the Cable Polarity IOCTL services. Added the Cable Isolation IOCTL services. Added the Cable Polarity IOCTL services.
February 27, 2014	Updated to version 2.5.52.0. Updated the kernel support table.
January 9, 2014	Updated to version 2.4.51.1. Updated the kernel support table.

16AO16 Linux Device Driver User Manual

January 7, 2014	Updated to version 2.4.51.0.
November 8, 2013	Updated to version 2.4.48.0.
July 3, 2013	Updated to version 2.4.45.0. Updated the kernel support table.
September 5, 2012	Updated to version 2.4.39.1.
July 18, 2012	Updated to version 2.4.39.0. Updated the kernel support table.
December 12, 2011	Updated to version 2.3.34.0.
October 31, 2011	Updated to version 2.2.30.0. Various editorial changes. Removed the IRQ_ENABLE and IRQ_STATUS IOCTL services. Renamed the IRQ_SEL IOCTL service values to IRQ. Updated the CPU and Kernel Support information. Updated the comments for the Initialize IOCTL service. Changed the spelling of various Autocalibration related software items.
December 24, 2009	Updated to version 2.1.13.0.
November 16, 2009	Updated to version 2.1.11.0. Corrected minor document errors. Changed AO16_IOCTL_BUFFER_STATE to AO16_IOCTL_BUFFER_STATUS. Added more sample applications.
November 4, 2009	Updated to version 2.0.10.0. The interface was overhauled.
September 29, 2008	Updated to version 1.3.0.
February 6, 2007	Updated to version 1.2.0.
July 1, 2005	Initial release.