

16AO64C

16-bit, 64/32/16 channel, 500K S/S/Ch Analog Output

PCIe-16AO64C

Linux Device Driver And API Library User Manual

**Manual Revision: August 5, 2024
Driver Release Version 2.10.111.50.0**

**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 © 2013-2024, **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.....	7
1.1. Purpose.....	7
1.2. Acronyms.....	7
1.3. Definitions	7
1.4. Software Overview	7
1.4.1. Basic Software Architecture	7
1.4.2. API Library.....	8
1.4.3. Device Driver	8
1.5. Hardware Overview	8
1.6. Reference Material.....	8
1.7. Licensing.....	9
2. Installation	10
2.1. CPU and Kernel Support.....	10
2.1.1. 32-bit Support Under 64-bit Environments	11
2.2. The /proc/ File System	11
2.3. File List.....	11
2.4. Directory Structure.....	11
2.5. Installation	12
2.6. Removal.....	12
2.7. Overall Make Script.....	12
2.8. Environment Variables	13
2.8.1. GSC_API_COMP_FLAGS.....	13
2.8.2. GSC_API_LINK_FLAGS.....	13
2.8.3. GSC_LIB_COMP_FLAGS.....	13
2.8.4. GSC_LIB_LINK_FLAGS.....	14
2.8.5. GSC_APP_COMP_FLAGS.....	14
2.8.6. GSC_APP_LINK_FLAGS.....	14
3. Main Interface Files.....	15
3.1. Main Header File	15
3.2. Main Library File.....	15
3.2.1. Build	15
3.2.2. System Libraries.....	16
3.2.3. Shared Object Script: Build the Main Libraries as Shard Object Files.....	16
4. API Library	17
4.1. Files.....	17
4.2. Build	17
4.3. Library Use	17
4.4. Macros	17

4.4.1. IOCTL Services	18
4.4.2. Registers	18
4.5. Data Types	18
4.6. Functions	18
4.6.1. ao64c_close()	19
4.6.2. ao64c_init()	19
4.6.3. ao64c_ioctl()	20
4.6.4. ao64c_open()	21
4.6.5. ao64c_read()	22
4.6.6. ao64c_write()	23
4.7. IOCTL Services	24
4.7.1. O64C_IOCTL_AUTOCAL	24
4.7.2. O64C_IOCTL_AUTOCAL_STATUS	24
4.7.3. O64C_IOCTL_BUFFER_CLEAR	25
4.7.4. O64C_IOCTL_BUFFER_LEVEL	25
4.7.5. O64C_IOCTL_BUFFER_MODE	25
4.7.6. O64C_IOCTL_BUFFER_OVER_DATA	25
4.7.7. O64C_IOCTL_BUFFER_OVER_FRAME	26
4.7.8. O64C_IOCTL_BUFFER_STATUS	26
4.7.9. O64C_IOCTL_BUFFER_THRESH_LVL	26
4.7.10. O64C_IOCTL_BUFFER_THRESH_STS	27
4.7.11. O64C_IOCTL_BURST_ENABLE	27
4.7.12. O64C_IOCTL_BURST_READY	27
4.7.13. O64C_IOCTL_BURST_TRIGGER	28
4.7.14. O64C_IOCTL_BURST_TRIG_SRC	28
4.7.15. O64C_IOCTL_CLOCK_MODE	28
4.7.16. O64C_IOCTL_CLOCK_READY	28
4.7.17. O64C_IOCTL_CLOCK_SRC	29
4.7.18. O64C_IOCTL_CLOCK_SW	29
4.7.19. O64C_IOCTL_DATA_FORMAT	29
4.7.20. O64C_IOCTL_GROUND_SENSE	30
4.7.21. O64C_IOCTL_INITIALIZE	30
4.7.22. O64C_IOCTL_IRQ_SEL	30
4.7.23. O64C_IOCTL_LOAD_READY	31
4.7.24. O64C_IOCTL_LOAD_REQUEST	31
4.7.25. O64C_IOCTL_OUTPUT_DISCONNECT	31
4.7.26. O64C_IOCTL_QUERY	32
4.7.27. O64C_IOCTL_RAG_ENABLE	33
4.7.28. O64C_IOCTL_RAG_NRATE	33
4.7.29. O64C_IOCTL_RANGE	33
4.7.30. O64C_IOCTL_RBG_ENABLE	34
4.7.31. O64C_IOCTL_RBG_NRATE	34
4.7.32. O64C_IOCTL_REG_MOD	34
4.7.33. O64C_IOCTL_REG_READ	35
4.7.34. O64C_IOCTL_REG_WRITE	35
4.7.35. O64C_IOCTL_TRIGGER_MODE	36
4.7.36. O64C_IOCTL_TX_IO_ABORT	36
4.7.37. O64C_IOCTL_TX_IO_BYTE_SWAP	36
4.7.38. O64C_IOCTL_TX_IO_MODE	37
4.7.39. O64C_IOCTL_TX_IO_OVER_DATA	37
4.7.40. O64C_IOCTL_TX_IO_OVER_FRAME	37
4.7.41. O64C_IOCTL_TX_IO_TIMEOUT	38
4.7.42. O64C_IOCTL_WAIT_CANCEL	38
4.7.43. O64C_IOCTL_WAIT_EVENT	39

4.7.44. O64C_IOCTL_WAIT_STATUS	41
5. The Driver.....	42
5.1. Files.....	42
5.2. Build	42
5.3. Startup.....	42
5.3.1. Manual Driver Startup Procedures	42
5.3.2. Automatic Driver Startup Procedures.....	43
5.4. Verification	44
5.5. Version.....	45
5.6. Shutdown	45
6. Document Source Code Examples.....	46
6.1. Files.....	46
6.2. Build	46
6.3. Library Use	46
7. Utilities Source Code.....	47
7.1. Files.....	47
7.2. Build	47
7.3. Library Use	47
8. Operating Information	48
8.1. Debugging Aids	48
8.1.1. Device Identification	48
8.1.2. Detailed Register Dump	48
8.2. Analog Output Configuration	48
8.3. Data Transfer Modes.....	48
8.3.1. PIO - Programmed I/O	49
8.3.2. BMDMA - Block Mode DMA	49
8.3.3. DMDMA - Demand Mode DMA	49
8.4. Multi-Board Synchronization	49
9. Sample Applications	50
9.1. aout - Analog Output - .../aout/	50
9.2. fsamp - Sample Rate - .../fsamp/	50
9.3. id - Identify Board - .../id/	50
9.4. mbsync - Multi-Board Synchronization - .../mbsync/	50
9.5. regs - Register Access - .../regs/	50
9.6. txrate - Transmit Rate - .../txrate/	50
Document History	51

Table of Figures

Figure 1 Basic architectural representation.....	8
Figure 2 The recommended configuration is a modified daisy-chain setup.	49

1. Introduction

1.1. Purpose

The purpose of this document is to describe the interface to the 16AO64C 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 16AO64C 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 16AO64C installation directory or any of its subdirectories.
16AO64C	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 16AO64C hardware.
Application	This is a user mode process, which runs in user space with user mode privileges.
Driver	This is the 16AO64C 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 16AO64C applications. The overall architecture is illustrated in Figure 1 below.

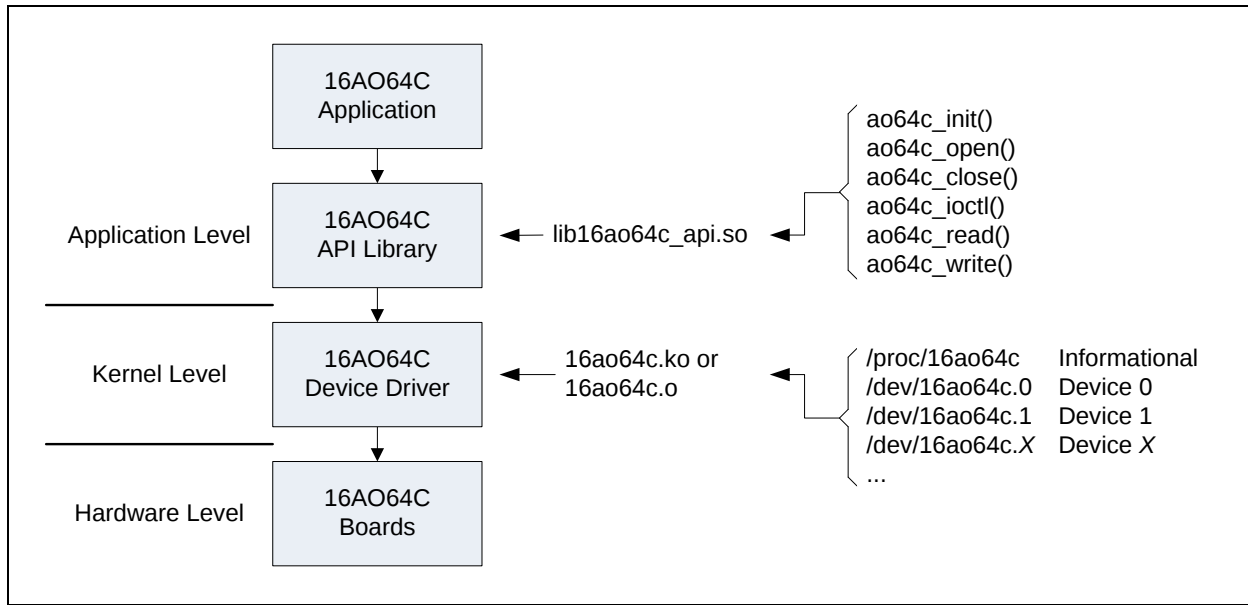


Figure 1 Basic architectural representation.

1.4.2. API Library

The primary means of accessing 16AO64C boards is via the 16AO64C API Library. This library forms a layer between the application and the driver. Additional information is given in section 3.2.3 (page 16). With the library, applications are able to open and close a device and, while open, perform I/O control and read operations.

1.4.3. Device Driver

The device driver is the host software that provides a means of communicating directly with 16AO64C 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 16AO64C is a high-performance, 16-bit analog output board that incorporates 64 or 32 Single Ended output channels, or 32 or 16 Differential output channels. The host side connection is PCI based and the form factor is according to the model ordered. The board is capable of outputting data at up to 500K samples per second per channel when using 32 or fewer channels. The maximum sample rate is 330K samples per channel when using 64 channels. (When using between 32 and 64 channels, the maximum sample rate is proportioned per the number of channels used.) Internal clocking permits sampling rates from 500K samples per second down to three samples per second. Onboard storage permits data buffering of up to 256K samples, for all channels collectively, between the PCI bus and the cable interface. This allows the 16AO64C to sustain continuous throughput to the cable interface independent of the PCI bus interface. The 16AO64C also permits multiple boards to be synchronized so that all boards output data in unison.

1.6. Reference Material

The following reference material may be of particular benefit in using the 16AO64C. The specifications provide the information necessary for an in depth understanding of the specialized features implemented on this board.

- The applicable *16AO64C 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
6.2.9	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/16ao64c` file will be "no".

2.2. The `/proc/` File System

While the driver is running, the text file `/proc/16ao64c` 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: 2.10.111.50
32-bit support: yes
boards: 1
models: 16AO64C
```

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>16ao64c.linux.tar.gz</code>	This archive contains the driver, the API Library and all related files.
<code>16ao64c_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>16ao64c/</code>	This is the driver root directory. It contains the documentation, the Overall Make Script (section 2.7, page 12) and the below listed subdirectories.
<code>.../api/</code>	This directory contains the API Library source files (section 3.2.3, page 16).
<code>.../docsrc/</code>	This directory contains the source files for the code samples given in this document (section 6, page 46).
<code>.../driver/</code>	This directory contains the device driver source files (section 5, page 42).
<code>.../include/</code>	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 50).
.../utils/	This directory contains the source files for the utility libraries used by the sample applications (section 7, page 47).

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 `16ao64c.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 `16ao64c` in the current directory, and then copies all of the archive's files into this new directory.

```
tar -xzvf 16ao64c.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 45).
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 16ao64c.linux.tar.gz 16ao64c
```

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

```
rm -f /dev/16ao64c.*
```

5. If the automatic startup procedure was adopted (section 5.3.2, page 43), then edit the system startup script `rc.local` and remove the line that invokes the 16AO64C'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 (`.../16ao64c/`).

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/lib16ao64c_api.so
Defined and Not Empty	==== Linking: ../lib/lib16ao64c_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/16ao64c_utils.a
Defined and Not Empty	==== Linking: ../lib/16ao64c_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 16AO64C 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 16AO64C 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 16AO64C specific header files. Therefore, sources may include only this one 16AO64C header and make files may reference only this one 16AO64C include directory.

Description	File	Location
Header File	16ao64c_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 16AO64C 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 16AO64C static library and only this one 16AO64C library directory.

Description	File	Location
Static Library	16ao64c_main.a	.../lib/
	16ao64c_multi.a	

NOTE: For applications using the 16AO64C and no other GSC devices, link the 16ao64c_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 16AO64C API Library is implemented as a shared library and is thus not linked with the 16AO64C Main Library. The API Library must be linked with applications by adding the argument -l16ao64c_api to the linker command line.

3.2.1. Build

The main library is built via the Overall Make Script (section 2.7, page 12). 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 12) 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	lib16ao64c_main.so	.../lib/
	lib16ao64c_multi.so	
	lib16ao64c_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 16AO64C 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
lib16ao64c_main.so	-l16ao64c_main
lib16ao64c_multi.so	-l16ao64c_multi
lib16ao64c_all.so†	-l16ao64c_all

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

4. API Library

The 16AO64C API Library is the software interface between user applications and the 16AO64C device driver. The interface is accessed by including the header file `16ao64c_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	<code>16ao64c_api.h</code>	.../include/
Library File	<code>lib16ao64c_api.so</code>	.../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 12), 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	<code>16ao64c_api.h</code>	.../include/	
Shared Object Library	<code>lib16ao64c_api.so</code>	.../lib/ /usr/lib/	<code>-l16ao64c_api</code>

4.4. Macros

The API Library and driver interfaces include the following macros, which are defined in `16ao64c.h`.

4.4.1. IOCTL Services

The IOCTL macros are documented in section 4.7 (page 24).

4.4.2. Registers

The following gives the complete set of 16AO64C registers.

4.4.2.1. GSC Registers

The following table gives the complete set of GSC specific 16AO64C 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 *16AO64C 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
O64C_GSC_ACR	Assembly Configuration Register (ACR)
O64C_GSC_AVR	Autocal Values Register (AVR)
O64C_GSC_BCR	Board Control Register (BCR)
O64C_GSC_BOR	Buffer Operations Register (BOR)
O64C_GSC_BSR	Buffer Size Register (BSR)
O64C_GSC_BTR	Buffer Threshold Register (BTR)
O64C_GSC_ODBR	Output Data Buffer Register (ODBR)
O64C_GSC_RAGR	Rate-A Generator Register (RAGR)
O64C_GSC_RBGR	Rate-B Generator Register (RBGR)

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 `16ao64c_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 `16ao64c_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 24).

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. ao64c_close()

This function is the entry point to close a connection made via the API's open call (section 4.6.4, page 21). The device is put in an initialized state before this call returns.

Prototype

```
int ao64c_close(int fd);
```

Argument	Description
fd	This is the file descriptor obtained from the open service (section 4.6.4, page 21).

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

Example

```
#include <stdio.h>

#include "16ao64c_dsl.h"

int ao64c_close_dsl(int fd)
{
    int errs;
    int ret;

    ret = ao64c_close(fd);

    if (ret)
        printf("ERROR: ao64c_close() returned %d\n", ret);

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

4.6.2. ao64c_init()

This function is the entry point to initializing the 16AO64C 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 ao64c_init(void);
```

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

Example

```

#include <stdio.h>

#include "16ao64c_dsl.h"

int ao64c_init_dsl(void)
{
    int errs;
    int ret;

    ret = ao64c_init();

    if (ret)
        printf("ERROR: ao64c_init() returned %d\n", ret);

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

```

4.6.3. ao64c_ioctl()

This function is the entry point to performing setup and control operations on a 16AO64C. 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 24).

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

Prototype

```
int ao64c_ioctl(int fd, int request, void* arg);
```

Argument	Description
<code>fd</code>	This is the file descriptor obtained from the open service (section 4.6.4, page 21).
<code>request</code>	This specifies the desired operation to be performed (section 4.7, page 24).
<code>arg</code>	This is specific to the IOCTL operation specified by the <code>request</code> argument.

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

Example

```

#include <stdio.h>

#include "16ao64c_dsl.h"

int ao64c_ioctl_dsl(int fd, int request, void* arg)
{
    int errs;
    int ret;

```

```

ret = ao64c_ioctl(fd, request, arg);

if (ret)
    printf("ERROR: ao64c_ioctl() returned %d\n", ret);

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

```

4.6.4. ao64c_open()

This function is the entry point to open a connection to a 16AO64C board. Before returning, the initialize IOCTL service is called to reset all hardware and software settings to their defaults.

Prototype

```
int ao64c_open(int device, int share, int* fd);
```

Argument	Description						
device	This is the zero-based index of the 16AO64C to access. †						
share	Open the device in Shared Access Mode? If non-zero the device is opened in Shared Access Mode (see below). If zero the device is opened in Exclusive Access Mode (see below).						
fd	The device handle is returned here. The pointer cannot be NULL. Values returned are as follows. <table border="1"> <tr> <th>Value</th><th>Description</th></tr> <tr> <td>>= 0</td><td>This is the handle to use to access the device in subsequent calls.</td></tr> <tr> <td>-1</td><td>There was an error. The device is not accessible.</td></tr> </table>	Value	Description	>= 0	This is the handle to use to access the device in subsequent calls.	-1	There was an error. The device is not accessible.
Value	Description						
>= 0	This is the handle to use to access the device in subsequent calls.						
-1	There was an error. The device is not accessible.						

† The index value -1 can also be given to acquire driver information (section 2.2, page 11).

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

Example

```

#include <stdio.h>

#include "16ao64c_dsl.h"

int ao64c_open_dsl(int device, int share, int* fd)
{
    int errs;
    int ret;

    ret = ao64c_open(device, share, fd);

    if (ret)
        printf("ERROR: ao64c_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. `ao64c_read()`

This function is the entry point to reading data from an open connection. The function reads up to `bytes` bytes.

NOTE: If an open was performed using an index of `-1`, then read requests will acquire information from the driver (section 2.2, page 11) rather than data from a device.

NOTE: The read service has no functionality for reading from 16AO64C devices. Attempts to read from 16AO64C devices will return an error.

Prototype

```
int ao64c_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 21).
<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 to bytes</code>	The operation succeeded.
<code>< 0</code>	An error occurred. See error value description above.

Example

```
#include <stdio.h>

#include "16ao64c_dsl.h"

int ao64c_read_dsl(int fd, void* dst, size_t bytes, size_t* qty)
{
    int errs;
    int ret;

    ret = ao64c_read(fd, dst, bytes);
```

```

    if (ret < 0)
        printf("ERROR: ao64c_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. ao64c_write()

This function is the entry point to writing data to an open 16AO64C. This function should only be called after a successful open of the respective device. The function writes up to `bytes` bytes to the board. The return value is the number of bytes actually written.

NOTE: When performing an open on device index -1, the API Library accesses the `/proc/16ao64c` text file. In this instance, all write requests will fail.

NOTE: The format of the 32-bit data values changed as of firmware version 0x400. If the board firmware version is 0x400 or higher, then the control fields are in the upper byte. If the firmware version is less than 0x400, then the control fields are in bits D23 to D16. The driver will move the control fields to the correct location, if called for. However, it is much more efficient for applications to generate correctly formatted data than it is for the driver make the correction. Please see the `O64C_IOCTL_TX_IO_BYTE_SWAP` service for additional information (section 4.7.37, page 36).

Prototype

```
int ao64c_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 21).
<code>src</code>	The data written comes from here.
<code>bytes</code>	This is the desired number of bytes to write.

Return Value	Description
0 to <code>bytes</code>	The operation succeeded. A value less than <code>bytes</code> indicates that the request timed out.
< 0	An error occurred. See error value description above.

Example

```

#include <stdio.h>

#include "16ao64c_dsl.h"

int ao64c_write_dsl(int fd, const void* src, size_t bytes, size_t*
qty)
{
    int errs;
    int ret;

    ret = ao64c_write(fd, src, bytes);

```

```

    if (ret < 0)
        printf("ERROR: ao64c_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 16AO64C API Library and device driver implement the following IOCTL services. Each service is described along with the applicable `ao64c_ioctl()` function arguments.

4.7.1. O64C_IOCTL_AUTOCAL

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

WARNING: Do not access the board while an autocalibration cycle is in progress. Doing so may produce indeterminate results, and may lockup the board.

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	O64C_IOCTL_AUTOCAL
arg	Not used.

4.7.2. O64C_IOCTL_AUTOCAL_STATUS

This service retrieves the autocalibration completion status.

Usage

Argument	Description
request	O64C_IOCTL_AUTOCAL_STATUS
arg	s32*

The value returned will be one of the following.

Value	Description
O64C_AUTOCAL_STATUS_ACTIVE	Autocalibration is in progress.
O64C_AUTOCAL_STATUS_FAIL	Autocalibration failed.
O64C_AUTOCAL_STATUS_PASS	Autocalibration passed.

4.7.3. O64C_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 data output. The driver waits for up to the current I/O write timeout for completion, before returning.

Usage

Argument	Description
request	O64C_IOCTL_BUFFER_CLEAR
arg	Not used.

4.7.4. O64C_IOCTL_BUFFER_LEVEL

This service retrieves the current fill level for the output buffer.

Usage

Argument	Description
request	O64C_IOCTL_BUFFER_LEVEL
arg	s32*

The value returned will be from zero to 0x40000.

4.7.5. O64C_IOCTL_BUFFER_MODE

This service configures the board's handling of data once it leaves the output buffer.

Usage

Argument	Description
request	O64C_IOCTL_BUFFER_MODE
arg	s32*

Valid argument values are as follows.

Value	Description
-1	Retrieve the current setting.
O64C_BUFFER_MODE_CIRC	Data is recycled when it exits the buffer.
O64C_BUFFER_MODE_OPEN	Data is not recycled when it exits the buffer.

4.7.6. O64C_IOCTL_BUFFER_OVER_DATA

This service operates on the Buffer Overflow status.

Usage

Argument	Description
request	O64C_IOCTL_BUFFER_OVER_DATA
arg	s32*

Valid argument values are as follows.

Value	Description
O64C_BUFFER_OVERFLOW_CHK	Report if an overflow has occurred.
O64C_BUFFER_OVERFLOW_CLR	Clear the overflow status.

The following values are those returned when checking on the overflow status.

Value	Description
O64C_BUFFER_OVERFLOW_NO	An overflow did not occur.
O64C_BUFFER_OVERFLOW_YES	An overflow did occur.

4.7.7. O64C_IOCTL_BUFFER_OVERFLOW_FRAME

This service operates on the Frame Overflow status.

Usage

Argument	Description
request	O64C_IOCTL_BUFFER_OVERFLOW_FRAME
arg	s32*

Valid argument values are as follows.

Value	Description
O64C_BUFFER_OVERFLOW_FRAME_CHK	Report if an overflow has occurred.
O64C_BUFFER_OVERFLOW_FRAME_CLR	Clear the overflow status.

The following values are those returned when checking on the overflow status.

Value	Description
O64C_BUFFER_OVERFLOW_FRAME_NO	An overflow did not occur.
O64C_BUFFER_OVERFLOW_FRAME_YES	An overflow did occur.

4.7.8. O64C_IOCTL_BUFFER_STATUS

This service reports the relative fill level of the output buffer.

Usage

Argument	Description
request	O64C_IOCTL_BUFFER_STATUS
arg	s32*

The service returns one of the following values.

Value	Description
O64C_BUFFER_STATUS_EMPTY	The buffer is empty.
O64C_BUFFER_STATUS_LOW	The buffer contains Threshold Level or fewer samples.
O64C_BUFFER_STATUS_HIGH	The buffer contains more than Threshold Level samples.
O64C_BUFFER_STATUS_FULL	The buffer is full.

4.7.9. O64C_IOCTL_BUFFER_THRESH_LVL

This service accesses the Buffer Threshold Level, which is the buffer fill level associated with the Buffer Threshold Status.

Usage

Argument	Description
request	O64C_IOCTL_BUFFER_THRESH_LVL
arg	s32*

Valid argument values are from zero to 0x7FFF, or -1 to retrieve the current setting.

4.7.10. O64C_IOCTL_BUFFER_THRESH_STS

This service operates on the Frame Overflow status.

Usage

Argument	Description
request	O64C_IOCTL_BUFFER_THRESH_STS
arg	s32*

The service returns one of the following values.

Value	Description
O64C_BUFFER_THRESH_STS_CLR	The buffer fill level is at or below the current Buffer Threshold Level (section 4.7.9, page 26).
O64C_BUFFER_THRESH_STS_SET	The buffer fill level exceeds the current Buffer Threshold Level (section 4.7.9, page 26).

4.7.11. O64C_IOCTL_BURST_ENABLE

This service enables or disables output bursting.

Usage

Argument	Description
request	O64C_IOCTL_BURST_ENABLE
arg	s32*

Valid argument values are as follows.

Value	Description
-1	Retrieve the current state.
O64C_BURST_ENABLE_NO	This refers to output bursting being disabled.
O64C_BURST_ENABLE_YES	This refers to output bursting being enabled.

4.7.12. O64C_IOCTL_BURST_READY

This service reports the board's readiness for burst initiation.

Usage

Argument	Description
request	O64C_IOCTL_BURST_READY
arg	s32*

The service returns one of the following values.

Value	Description
O64C_BURST_READY_NO	The board is not ready for burst initiation.
O64C_BURST_READY_YES	The board is ready for burst initiation.

4.7.13. O64C_IOCTL_BURST_TRIGGER

This service initiates an output burst cycle. The service waits for up to the write timeout period for the operation to complete.

Usage

Argument	Description
request	O64C_IOCTL_BURST_TRIGGER
arg	Not used.

4.7.14. O64C_IOCTL_BURST_TRIG_SRC

This service controls the trigger source selection for triggered burst operation.

Usage

Argument	Description
request	O64C_IOCTL_BURST_TRIG_SRC
arg	s32*

Valid argument values are as follows.

Value	Description
-1	Retrieve the current setting.
O64C_BURST_TRIG_SRC_EXT	Utilize external burst triggering.
O64C_BURST_TRIG_SRC_RBG	Utilize the output from the Rate-B Generator.

4.7.15. O64C_IOCTL_CLOCK_MODE

This service configures the operating mode for the cable's clock signal.

Usage

Argument	Description
request	O64C_IOCTL_CLOCK_MODE
arg	s32*

Valid argument values are as follows.

Value	Description
-1	Retrieve the current setting.
O64C_CLOCK_MODE_INITIATOR	The cable clock pin is an output.
O64C_CLOCK_MODE_TARGET	The cable clock pin is an input.

4.7.16. O64C_IOCTL_CLOCK_READY

This service reports the board's readiness to respond to clocking.

Usage

Argument	Description
request	O64C_IOCTL_CLOCK_READY
arg	s32*

The service returns one of the following values.

Value	Description
O64C_CLOCK_READY_NO	The board is not ready for clocking.
O64C_CLOCK_READY_YES	The board is ready for clocking.

4.7.17. O64C_IOCTL_CLOCK_SRC

This service selects the source for the output sample clock.

Usage

Argument	Description
request	O64C_IOCTL_CLOCK_SRC
arg	s32*

Valid argument values are as follows.

Value	Description
-1	Retrieve the current setting.
O64C_CLOCK_SRC_EXT	This selects the cable interface as the clock source.
O64C_CLOCK_SRC_RAG	This selects the Rate-A Generator as the clock source.

4.7.18. O64C_IOCTL_CLOCK_SW

This service initiates an output clock cycle. The service waits for up to the write timeout period for the operation to complete.

Usage

Argument	Description
request	O64C_IOCTL_CLOCK_SW
arg	Not used.

4.7.19. O64C_IOCTL_DATA_FORMAT

This service sets the data encoding format.

Usage

Argument	Description
request	O64C_IOCTL_DATA_FORMAT
arg	s32*

Valid argument values are as follows.

Value	Description
-1	Retrieve the current setting.

O64C_DATA_FORMAT_2S_COMP	Select the Twos Compliment data format.
O64C_DATA_FORMAT_OFF_BIN	Select the Offset Binary encoding format.

4.7.20. O64C_IOCTL_GROUND_SENSE

This service configures the board's ground sense logic.

Usage

Argument	Description
request	O64C_IOCTL_GROUND_SENSE
arg	s32*

Valid argument values are as follows.

Value	Description
-1	Retrieve the current setting.
O64C_GROUND_SENSE_DISABLE	This disables remote ground sensing.
O64C_GROUND_SENSE_REMOTE	This selects remote ground sensing.

4.7.21. O64C_IOCTL_INITIALIZE

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

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

Usage

Argument	Description
request	O64C_IOCTL_INITIALIZE
arg	Not used.

4.7.22. O64C_IOCTL_IRQ_SEL

This service configures the interrupt source selection for the firmware interrupt.

Usage

Argument	Description
request	O64C_IOCTL_IRQ_SEL
arg	s32*

Valid argument values are as follows.

Value	Description
-1	Retrieve the current setting.
O64C_IRQ_AUTOCAL_DONE	This refers to the completion of autocalibration.
O64C_IRQ_BUF_EMPTY	This refers to the buffer becoming empty.
O64C_IRQ_BUF_THRESH_HI2LO	This refers to the buffer fill level dropping to the current Threshold Level.
O64C_IRQ_BUF_THRESH_LO2HI	This refers to the buffer fill level rising to exceed the current Threshold Level.
O64C_IRQ_BURST_TRIG_READY	This refers to the board becoming ready for a burst trigger.

O64C_IRQ_INIT_DONE	This refers to the completion of initialization.
O64C_IRQ_LOAD_READY	This refers to a circular buffer becoming ready to receive data.
O64C_IRQ_LOAD_READY_END	This refers to a circular buffer becoming not ready to receive data.

4.7.23. O64C_IOCTL_LOAD_READY

This service reports the buffer's readiness to receive additional data when in circular buffer mode.

Usage

Argument	Description
request	O64C_IOCTL_LOAD_READY
arg	s32*

Valid values returned by the service are as follows.

Value	Description
O64C_LOAD_READY_NO	The buffer is not ready to receive additional data.
O64C_LOAD_READY_YES	The buffer is ready to receive additional data.

4.7.24. O64C_IOCTL_LOAD_REQUEST

This service requests that buffer become ready to receive additional data when in circular buffer mode. The service waits for up to the write timeout period for the operation to complete.

Usage

Argument	Description
request	O64C_IOCTL_LOAD_REQUEST
arg	Not used.

4.7.25. O64C_IOCTL_OUTPUT_DISCONNECT

This service configures the board logic that disconnects the output signals from the cable interface.

NOTE: This service is operable only if the disconnect feature is support by the board. Refer to the O64C_QUERY_DISCONNECT query option (section 4.7.26, page 32).

Usage

Argument	Description
request	O64C_IOCTL_OUTPUT_DISCONNECT
arg	s32*

Valid argument values are as follows.

Value	Description
-1	Retrieve the current setting.
O64C_OUTPUT_DISCONNECT_NO	This disables remote ground sensing.
O64C_OUTPUT_DISCONNECT_YES	This selects remote ground sensing. This option is available only if the board supports the disconnect feature.

4.7.26. O64C_IOCTL_QUERY

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

Usage

Argument	Description
request	O64C_IOCTL_QUERY
arg	s32*

Valid argument values are as follows.

Value	Description
O64C_QUERY_AUTOCAL_MS	This returns the maximum duration of the Autocalibration cycle in milliseconds.
O64C_QUERY_CHANNEL_MAX	This returns the maximum number of output channels supported by all boards of the same model as the board accessed.
O64C_QUERY_CHANNEL_QTY	This returns the actual number of output channels on the current board.
O64C_QUERY_COUNT	This returns the number of query options supported by the IOCTL service.
O64C_QUERY_DEVICE_TYPE	This returns the identifier value for the board's type. The value should be GSC_DEV_TYPE_16AO64C.
O64C_QUERY_DIFFERENTIAL	This returns an indication if the board is configured for Differential operation. Non-zero means yes. Zero means Single Ended.
O64C_QUERY_DISCONNECT	This refers to the board feature that is able to disconnect the analog output signals from the cable interface.
O64C_QUERY_FIFO_SIZE	This returns the size of the output buffer in samples.
O64C_QUERY_FILTER	This returns the identifier for the installed filter option. Refer to the table below for the returned option values.
O64C_QUERY_FREF_DEFAULT	This gives the default FREF value in hertz.
O64C_QUERY_FSAMP_MAX	This gives the maximum sample rate in S/S.
O64C_QUERY_FSAMP_MIN	This gives the minimum sample rate in S/S.
O64C_QUERY_INIT_MS	This returns the duration of a board initialization in milliseconds.
O64C_QUERY_LAST	This is included for reference only and should not be used by applications. Applications should use the COUNT option instead.
O64C_QUERY_NRATE_MASK	This returns the mask for the rate generator NRATE fields.
O64C_QUERY_NRATE_MAX	This returns the maximum supported NRATE value.
O64C_QUERY_NRATE_MIN	This returns the minimum supported NRATE value.
O64C_QUERY_RANGE	This returns the identifier for the installed voltage range option. Refer to the table below for the returned option values.

Valid return values for the O64C_QUERY_FILTER option are as follows.

Value	Description
O64C_FILTER_NONE	No filter is installed.
O64C_FILTER_10KHZ	A 10KHz filter is installed.
O64C_FILTER_100KHZ	A 100KHz filter is installed.

Valid return values for the O64C_QUERY_RANGE option are as follows.

Value	Description
O64C_RANGE_BI_10_5	The board is hardwired for the bipolar ranges of $\pm 10V$ and $\pm 5V$.
O64C_RANGE_UNI_10_5	The board is hardwired for the unipolar ranges of 0-10V and 0-5V.

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

Value	Description
O64C_IOCTL_QUERY_ERROR	Either there was a processing error or the query option is unrecognized.

4.7.27. O64C_IOCTL_RAG_ENABLE

This service enables or disables the Rate-A Generator.

Usage

Argument	Description
request	O64C_IOCTL_RAG_ENABLE
arg	s32*

Valid argument values are as follows.

Value	Description
-1	Retrieve the current state.
O64C_RAG_ENABLE_NO	This disables the generator.
O64C_RAG_ENABLE_YES	This enables the generator.

4.7.28. O64C_IOCTL_RAG_NRATE

This service sets the NRATE adjustment value for the Rate-A Generator.

Usage

Argument	Description
request	O64C_IOCTL_RAG_NRATE
arg	s32*

Valid argument values are in the range from 98 to 0xFFFFFFFF, and -1. The value -1 is used to retrieve the current setting.

4.7.29. O64C_IOCTL_RANGE

This service configures the analog output voltage range.

Usage

Argument	Description
request	O64C_IOCTL_RANGE
arg	s32*

Valid argument values are as follows.

Value	Description
-1	Retrieve the current setting.
O64C_RANGE_5V	Select the 5V output range. The actual range is ± 5 volts for Differential boards, or 0-5 volts for Single Ended boards.
O64C_RANGE_10V	Select the 10V output range. The actual range is ± 10 volts for Differential boards, or 0-10 volts for Single Ended boards.

4.7.30. O64C_IOCTL_RBG_ENABLE

This service enables or disables the Rate-B Generator.

Usage

Argument	Description
request	O64C_IOCTL_RBG_ENABLE
arg	s32*

Valid argument values are as follows.

Value	Description
-1	Retrieve the current state.
O64C_RBG_ENABLE_NO	This disables the generator.
O64C_RBG_ENABLE_YES	This enables the generator.

4.7.31. O64C_IOCTL_RBG_NRATE

This service sets the NRATE adjustment value for the Rate-B Generator.

Usage

Argument	Description
request	O64C_IOCTL_RBG_NRATE
arg	s32*

Valid argument values are in the range from 98 to 0xFFFFFFFF, and -1. The value -1 is used to retrieve the current setting.

4.7.32. O64C_IOCTL_REG_MOD

This service performs a read-modify-write of a 16AO64C register. This includes only the GSC firmware registers. The PCI and PLX Feature Set Registers are read-only. Refer to `16ao64c.h` for a complete list of the GSC firmware registers.

Usage

Argument	Description
request	O64C_IOCTL_REG_MOD
arg	gsc_reg_t*

Definition

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

Fields	Description
reg	This is set to the identifier for the register to access.
value	This contains the value for the register bits to modify.

mask	This specifies the set of bits to modify. If a bit here is set, then the respective register bits is modified. If a bit here is zero, then the respective register bit is unmodified.
------	---

4.7.33. O64C_IOCTL_REG_READ

This service reads the value of a 16AO64C register. This includes the PCI registers, the PLX Feature Set Registers and the GSC firmware registers. Refer to `16ao64c.h` and `gsc_pci9056.h` for the complete list of accessible registers.

Usage

Argument	Description
request	O64C_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.

4.7.34. O64C_IOCTL_REG_WRITE

This service writes a value to a 16AO64C register. This includes only the GSC firmware registers. The PCI and PLX Feature Set Registers are read-only. Refer to `16ao64c.h` for a complete list of the GSC firmware registers.

Usage

Argument	Description
request	O64C_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.

4.7.35. O64C_IOCTL_TRIGGER_MODE

This service configures the operating mode for the cable's trigger signal.

Usage

Argument	Description
request	O64C_IOCTL_TRIGGER_MODE
arg	s32*

Valid argument values are as follows.

Value	Description
-1	Retrieve the current setting.
O64C_TRIGGER_MODE_INITIATOR	The cable trigger pin is an output.
O64C_TRIGGER_MODE_TARGET	The cable trigger pin is an input.

4.7.36. O64C_IOCTL_TX_IO_ABORT

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

Usage

Argument	Description
request	O64C_IOCTL_TX_IO_ABORT
arg	s32*

The results are reported as one of the following values.

Value	Description
O64C_IO_ABORT_NO	A <code>write()</code> request was not aborted as none were ongoing.
O64C_IO_ABORT_YES	An ongoing <code>write()</code> request was aborted.

4.7.37. O64C_IOCTL_TX_IO_BYTE_SWAP

This service tells the driver what action to take with respect to making sure the data stream is properly formatted for the current board firmware. Please see the `write()` service for additional information (section 4.6.4.1, page 22).

Usage

Argument	Description
request	O64C_IOCTL_TX_IO_BYTE_SWAP
arg	s32*

Valid argument values are as follows.

Value	Description
-1	Retrieve the current setting.
O64C_TX_IO_BYTE_SWAP_CHECK	Examine the data to see if byte swapping is called for. If it is, then the error <code>EINVAL</code> is reported. †
O64C_TX_IO_BYTE_SWAP_DISABLE	Disable byte swap processing all together.
O64C_TX_IO_BYTE_SWAP_PERFORM	Examine the data and perform byte swapping is called for. The error <code>EINVAL</code> will be reported if the unused byte is non-zero. †

† When performed, the examination looks only for the first sample with a non-zero upper word. The examination is performed as each block or sub-block of data is transferred from the host to the board.

4.7.38. O64C_IOCTL_TX_IO_MODE

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

Usage

Argument	Description
request	O64C_IOCTL_TX_IO_MODE
arg	s32*

Valid argument values are as follows.

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

4.7.39. O64C_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 buffer overflow

Usage

Argument	Description
request	O64C_IOCTL_TX_IO_OVER_DATA
arg	s32*

Valid argument values are as follows.

Value	Description
-1	Retrieve the current setting.
O64C_TX_IO_OVER_DATA_CHECK	Perform the check. This is the default.
O64C_TX_IO_OVER_DATA_IGNORE	Do not perform the check.

4.7.40. O64C_IOCTL_TX_IO_OVER_FRAME

This service configures the write service to check for a frame overflow before performing write operations. Sample data is lost when there is a frame overflow

Usage

Argument	Description
request	O64C_IOCTL_TX_IO_OVER_FRAME
arg	s32*

Valid argument values are as follows.

Value	Description
-1	Retrieve the current setting.

O64C_TX_IO_OVER_FRAME_CHECK	Perform the check. This is the default.
O64C_TX_IO_OVER_FRAME_IGNORE	Do not perform the check.

4.7.41. O64C_IOCTL_TX_IO_TIMEOUT

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

Usage

Argument	Description
request	O64C_IOCTL_TX_IO_TIMEOUT
arg	s32*

Valid argument values are in the range from zero to 3600, -1, and O64C_IO_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 O64C_IO_TIMEOUT_INFINITE is used, then the driver will wait indefinitely rather than timing out. The default is 10 seconds.

4.7.42. O64C_IOCTL_WAIT_CANCEL

This service resumes all threads blocked via O64C_IOCTL_WAIT_EVENT IOCTL calls (section 4.7.43, page 39), 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	O64C_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 4.7.43.2 (page 40).
gsc	This specifies the set of O64C_WAIT_GSC_* events whose wait requests are to be cancelled. Refer to section 4.7.43.3 (page 40).
alt	This is unused by the 16AO64C driver and should be zero.

io	This specifies the set of O64C_WAIT_IO_* events whose wait requests are to be cancelled. Refer to section 4.7.43.4 (page 40).
timeout_ms	This is unused by wait cancel operations.
count	Upon return this indicates the number of waits that were cancelled.

4.7.43. O64C_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	O64C_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 4.7.43.1 (page 40).
main	This specifies any number of GSC_WAIT_MAIN_* events that the thread is to wait for. Refer to section 4.7.43.2 (page 40).
gsc	This specifies any number of O64C_WAIT_GSC_* events that the thread is to wait for. Refer to section 4.7.43.3 (page 40).
alt	This is unused by the 16AO64C driver and must be zero.
io	This specifies any number of O64C_WAIT_IO_* events that the thread is to wait for. Refer to section 4.7.43.4 (page 40).
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.

4.7.43.1. `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.

4.7.43.2. `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 16AO64C 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 16AO64C.
<code>GSC_WAIT_MAIN_PCI</code>	This refers to any interrupt generated by the 16AO64C.
<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.

4.7.43.3. `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 `O64C_IOCTL_IRQ_SEL` (section 4.7.22, page 30).

Value	Description
<code>O64C_WAIT_GSC_AUTOCAL_DONE</code>	This refers to the completion of autocalibration.
<code>O64C_WAIT_GSC_BUF_EMPTY</code>	This refers to the buffer becoming empty.
<code>O64C_WAIT_GSC_BUF_THRESH_HI2LO</code>	This refers to the buffer level dropping to the Buffer Threshold level.
<code>O64C_WAIT_GSC_BUF_THRESH_LO2HI</code>	This refers to the buffer level rising above the Buffer Threshold level.
<code>O64C_WAIT_GSC_BURST_TRIG_READY</code>	This refers to the board becoming ready for a burst trigger.
<code>O64C_WAIT_GSC_INIT_DONE</code>	This refers to the completion of initialization.
<code>O64C_WAIT_GSC_LOAD_READY</code>	This refers to a circular buffer becoming ready to receive data.
<code>O64C_WAIT_GSC_LOAD_READY_END</code>	This refers to a circular buffer becoming not ready to receive data.

4.7.43.4. `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>O64C_WAIT_IO_TX_ABORT</code>	This refers to write requests which have been aborted.
<code>O64C_WAIT_IO_TX_DONE</code>	This refers to write requests which have been satisfied.
<code>O64C_WAIT_IO_TX_ERROR</code>	This refers to write requests which end due to an error.
<code>O64C_WAIT_IO_TX_TIMEOUT</code>	This refers to write requests which end due to the timeout period lapse.

4.7.44. O64C_IOCTL_WAIT_STATUS

This service count all threads blocked via the O64C_IOCTL_WAIT_EVENT IOCTL service (section 4.7.43, page 39), 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	O64C_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 4.7.43.2 (page 40).
gsc	This specifies the set of O64C_WAIT_GSC_* events whose wait requests are to be counted. Refer to section 4.7.43.3 (page 40).
alt	This is unused by the 16AO64C driver and should be zero.
io	This specifies the set of O64C_WAIT_IO_* events whose wait requests are to be counted. Refer to section 4.7.43.4 (page 40).
timeout_ms	This is unused by wait status operations.
count	Upon return this indicates the number of waits that met any of the specified criteria.

5. The Driver

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

5.1. Files

The device driver files are summarized in the table below.

Description	Files	Location
Source Files	*.c, *.h ...	.../driver/
Header File	16ao64c.h	
Driver File	16ao64c.ko † 16ao64c.o ‡	

† This is for kernel versions 2.6 and later.

‡ This is for kernel versions 2.4 are 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 12), 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 16AO64C 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 `16ao64c` 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/16ao64c.*
```

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/16ao64c/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 `rxwxr-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 `rxwxr-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 `rxwxr-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/16ao64c` 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/16ao64c
```

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/16ao64c` 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 16ao64c
```

2. Verify that the driver module has been unloaded by issuing the below command. The module name `16ao64c` 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	16ao64c_dsl.h	.../include/
Library File	16ao64c_dsl.a	.../lib/

6.2. Build

The library is built via the Overall Make Script (section 2.7, page 12), 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 15).

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 `ao64c_open()` there is the utility file `open.c` containing the utility function `ao64c_open_util()`. The naming pattern is as follows: API function `ao64c_xxxx()`, utility file name `xxxx.c`, utility function `ao64c_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 `AO64C_IOCTL_QUERY` there is the utility file `util_query.c` containing the utility function `ao64c_query()`. The naming pattern is as follows: IOCTL code `AO64C_IOCTL_XXXX`, utility file name `util_xxxx.c`, utility function `ao64c_xxxx()`.

7.1. Files

The utility files are summarized in the table below.

Description	Files	Location
Source Files	*.c, *.h ...	.../utils/
Header File	16ao64c_utils.h	.../include/
Library Files	16ao64c_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 12), 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 15).

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 16AO64C. 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 GSC register dump will include details of each register field.

Description	File/Name	Location
Function	ao64c_reg_list()	Source File
Source File	util_reg.c	.../utils/
Header File	16ao64c_utils.h	.../include/
Library File	16ao64c_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. The referenced files are included via the Main Header and Main Library.

Item	Name/File	Location
Function	ao64c_config_ao()	Source File
Source File	util_config_ao.c	.../utils/
Header File	16ao64c_utils.h	.../include/
Library File	16ao64c_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 Block Mode DMA the driver initiates DMA transfers only after a sufficient volume of data has been received into the input buffer. In this mode the volume is sufficient when the input buffer content satisfies the request or when it meets or exceeds the threshold value. After that amount of data is in the input buffer the driver initiates a DMA then sleeps until the DMA Done interrupt is received. Using this DMA mode, a user request typically consists of numerous individual DMA transfers.

8.3.3. DMDMA - Demand Mode DMA

This DMA mode is similar to the block mode, except that the transfer is initiated immediately. Here however, the actual movement of data occurs as the data becomes available in the buffer instead of after it has been accumulated. Using this DMA mode, a user request typically consists of a single individual DMA transfer.

8.4. Multi-Board Synchronization

Multi-board synchronization is a feature of the 16AO64C that enables two or more boards to clock output data in lock-step. Exercising this feature requires the boards to operate synchronously from the same clock source. This is done using the clock and trigger signals on the cable interface. Due to the number of available cable interface signals the multi-board signal routing configuration options is limited. The recommended configuration is the modified daisy-chain setup shown in Figure 2 below. This setup is applicable for any number of boards. The 16AO64C initiator can drive a limited number of targets. (For the specific limitations refer to the board user manual.) If targets beyond that limit are required, then a clock driver board is required.

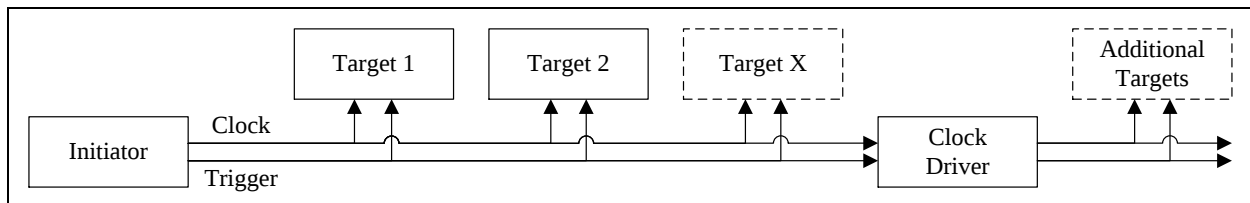


Figure 2 The recommended configuration is a modified daisy-chain setup.

The table below shows the board programming required for this configuration. The table shows the multi-board configuration specific settings applicable to the initiator and all connected targets. See the Multi-Board Synchronization sample application source code for additional programming requirements (section 9.4, page 50).

Setting	Initiator	Target(s)
Clock Mode	Initiator	Target
Trigger Mode	Initiator	Target
Clock Source	Rate-A Generator	External
Burst Trigger Source	Rate-B Generator	External

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 12), 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. fsamp - Sample Rate - .../fsamp/

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

9.3. 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.

9.4. mbsync - Multi-Board Synchronization - .../mbsync/

This application demonstrates multi-board configuration with two or more 16AO64C board wired together in a modified daisy-chain configuration as described in section 8.4, page 49.

9.5. regs - Register Access - .../regs/

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

9.6. 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
August 5, 2024	Updated to version 2.10.111.50.0. Numerous, minor editorial changes. Updated the kernel support table. Corrected release version numbers in this table.
May 19, 2023	Updated to version 2.9.104.47.0. Updated the kernel support table. Minor editorial changes. Updated the Output Buffer Clear service. Updated the description of the Autocalibration service. Renamed all autocalibration related macros.
October 7, 2022	Updated to version 2.8.101.43.0. Updated the information for the open and close calls.
July 13, 2022	Updated to version 2.8.100.42.0.
May 31, 2022	Updated to version 2.7.99.40.0. Expanded automatic startup information. Updated the kernel support table. Added section on environment variables. Various minor editorial updates.
October 28, 2020	Updated to version 2.6.91.35.0. Minor editorial changes.
October 23, 2020	Updated to version 2.6.85.27.1. User Manual update only. Updated the kernel support table. Various editorial updates. Added a note under the <code>ao64c_read()</code> service. Added a licensing subsection. Added WAIT_EVENT note. Expanded automatic startup information. Added multi-board synchronization information.
May 15, 2019	Updated to version 2.6.85.27.0. Minor editorial changes. Some file name changes. Updated the architectural representation figure.
September 4, 2018	Updated to version 2.5.80.26.0. Updated the inside cover page. Updated Block Mode DMA macro and associated information. Various other editorial changes.
March 19, 2018	Updated to version 2.4.76.21.0. Overhauled document. Implemented API Library, <code>include</code> directory and <code>lib</code> directory. Updated the CPU and kernel support section.
December 1, 2016	Updated to version 2.3.68.18.0. Updated the kernel support table. Added support for infinite I/O timeouts. Updated the operating information section. Made various miscellaneous updates. Some document reorganization.
July 20, 2016	Updated to version 2.2.67.15.0. Changed all <code>__s32</code> and <code>__u32</code> data types to <code>s32</code> and <code>u32</code> , respectively. Removed the <code>built</code> field from the <code>/proc/</code> file. Organized sample applications alphabetically. Updated the usage of the Wait Event <code>timeout_ms</code> field. Updated material on the open call. Added open access mode descriptions. Updated the kernel support table.
March 18, 2016	Updated to version 2.1.65.13.0. Change the <code>fsamp</code> sample rate command line argument to <code>-s#</code> .
March 3, 2014	Updated to version 1.2.52.0. Updated the kernel support table.
January 1, 2014	Updated to version 1.1.51.0. Updated the kernel support table.
November 15, 2013	Updated to version 1.1.50.0.
August 13, 2013	Updated to version 1.1.46.0. Updated the CPU support table. Added the Byte Swap IOCTL service (<code>O64C_IOCTL_TX_IO_BYTE_SWAP</code>). Updated the sample application command line information.
February 5, 2013	Initial release.