

16AO20

16-bit, 20/12/6 channel, 440K S/S/Ch Analog Output

PC104P-16AO20

Linux Device Driver And API Library User Manual

**Manual Revision: April 5, 2023
Driver Release Version 3.5.104.47.0**

**General Standards Corporation
8302A Whitesburg Drive
Huntsville, AL 35802
Phone: (256) 880-8787
Fax: (256) 880-8788**

URL: <http://www.generalstandards.com>

E-mail: sales@generalstandards.com

E-mail: support@generalstandards.com

Preface

Copyright © 2003-2023, **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
4. API Library	17
4.1. Files.....	17
4.2. Build	17
4.3. Library Use	17
4.4. Macros	18
4.4.1. IOCTL Services.....	18

4.4.2. Registers	18
4.5. Data Types	18
4.6. Functions.....	18
4.6.1. ao20_close()	19
4.6.2. ao20_init()	19
4.6.3. ao20_ioctl()	20
4.6.4. ao20_open().....	21
4.6.5. ao20_read().....	22
4.6.6. ao20_write()	23
4.7. IOCTL Services	24
4.7.1. AO20_IOCTL_AUTO_CAL_STS	24
4.7.2. AO20_IOCTL_AUTO_CALIBRATE	24
4.7.3. AO20_IOCTL_BUFFER_CLEAR.....	24
4.7.4. AO20_IOCTL_BUFFER_MODE	25
4.7.5. AO20_IOCTL_BUFFER_OVER_DATA	25
4.7.6. AO20_IOCTL_BUFFER_OVER_FRAME	25
4.7.7. AO20_IOCTL_BUFFER_SIZE	26
4.7.8. AO20_IOCTL_BUFFER_STATUS	26
4.7.9. AO20_IOCTL_BURST_ENABLE	27
4.7.10. AO20_IOCTL_BURST_READY	27
4.7.11. AO20_IOCTL_BURST_TRIG_SRC	27
4.7.12. AO20_IOCTL_BURST_TRIGGER.....	28
4.7.13. AO20_IOCTL_CHANNEL_SEL.....	28
4.7.14. AO20_IOCTL_CLOCK_ENABLE	28
4.7.15. AO20_IOCTL_CLOCK_OSC_SRC.....	28
4.7.16. AO20_IOCTL_CLOCK_READY	29
4.7.17. AO20_IOCTL_CLOCK_SRC.....	29
4.7.18. AO20_IOCTL_CLOCK_SW	29
4.7.19. AO20_IOCTL_DATA_FORMAT	30
4.7.20. AO20_IOCTL_EXT_CLK_IN_POL	30
4.7.21. AO20_IOCTL_EXT_TRIG_IN_POL	30
4.7.22. AO20_IOCTL_EXT_TRIG_OUT_POL	30
4.7.23. AO20_IOCTL_GROUND_SENSE	31
4.7.24. AO20_IOCTL_INITIALIZE.....	31
4.7.25. AO20_IOCTL_IRQ_SEL.....	31
4.7.26. AO20_IOCTL_LOAD_READY	32
4.7.27. AO20_IOCTL_LOAD_REQUEST.....	32
4.7.28. AO20_IOCTL_NCLK.....	32
4.7.29. AO20_IOCTL_NRATE	33
4.7.30. AO20_IOCTL_OUTPUT_MODE	33
4.7.31. AO20_IOCTL_QUERY	33
4.7.32. AO20_IOCTL_REG_MOD	35
4.7.33. AO20_IOCTL_REG_READ.....	35
4.7.34. AO20_IOCTL_REG_WRITE	36
4.7.35. AO20_IOCTL_SAMPLE_MODE	36
4.7.36. AO20_IOCTL_TX_IO_ABORT.....	36
4.7.37. AO20_IOCTL_TX_IO_MODE	37
4.7.38. AO20_IOCTL_TX_IO_OVER_DATA	37
4.7.39. AO20_IOCTL_TX_IO_OVER_FRAME.....	37
4.7.40. AO20_IOCTL_TX_IO_TIMEOUT	38
4.7.41. AO20_IOCTL_WAIT_CANCEL	38
4.7.42. AO20_IOCTL_WAIT_EVENT	39
4.7.43. AO20_IOCTL_WAIT_STATUS	41
4.7.44. AO20_IOCTL_XCVR_TYPE.....	41

5. The Driver.....	43
5.1. Files.....	43
5.2. Build	43
5.3. Startup.....	43
5.3.1. Manual Driver Startup Procedures	43
5.3.2. Automatic Driver Startup Procedures.....	44
5.4. Verification	45
5.5. Version.....	46
5.6. Shutdown	46
6. Document Source Code Examples.....	47
6.1. Files.....	47
6.2. Build	47
6.3. Library Use	47
7. Utility Source Code	48
7.1. Files.....	48
7.2. Build	48
7.3. Library Use	48
8. Operating Information	49
8.1. Analog Output Configuration	49
8.2. Debugging Aids	49
8.2.1. Device Identification	49
8.2.2. Detailed Register Dump	49
8.3. Data Transfer Modes.....	49
8.3.1. PIO - Programmed I/O	49
8.3.2. BMDMA - Block Mode DMA	50
9. Sample Applications	51
9.1. aout - Analog Output - ../aout/	51
9.2. fsamp - Sample Rate - ../fsamp/	51
9.3. id - Identify Board - ../id/	51
9.4. regs - Register Access - ../regs/	51
9.5. sbtest - Single Board Test - ../sbtest/	51
9.6. txrate - Transmit Rate - ../txrate/	51
Document History	52

Table of Figures

Figure 1 Basic architectural representation.....8

1. Introduction

1.1. Purpose

The purpose of this document is to describe the interface to the 16AO20 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 16AO20 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
DMA	Direct Memory Access
GSC	General Standards Corporation
PC104P	This refers to the PC/104+ form factor.
PCI	Peripheral Component Interconnect
PIO	Programmed I/O
PMC	PCI Mezzanine Card

1.3. Definitions

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

Term	Definition
...	This is a shortcut representation of the 16AO20 installation directory or any of its subdirectories.
16AO20	This is used as a general reference to any board supported by this driver.
API Library	This is a library that provides application-level access to 16AO20 hardware.
Application	This is a user mode process, which runs in user space with user mode privileges.
Driver	This is the 16AO20 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 16AO20 applications. The overall architecture is illustrated in Figure 1 below.

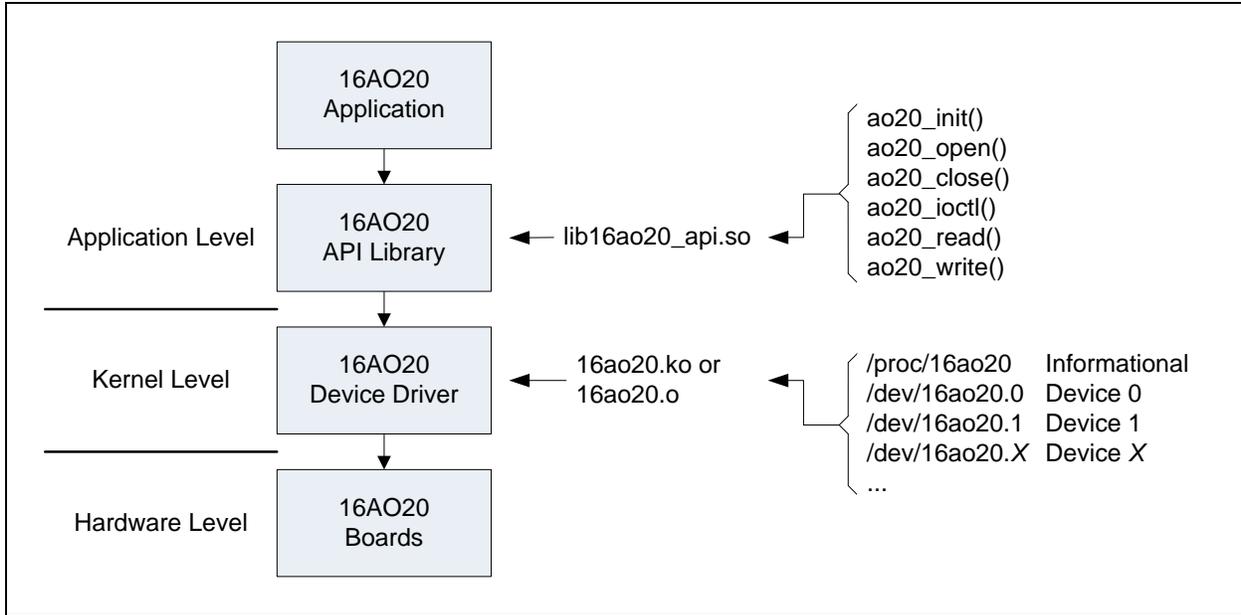


Figure 1 Basic architectural representation.

1.4.2. API Library

The primary means of accessing 16AO20 boards is via the 16AO20 API Library. This library forms a thin layer between the application and the driver. Additional information is given in section 4 beginning on page 17. 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 16AO20 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 16AO20 is a high-performance, 16-bit analog output board that incorporates 20, 12 or 6 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 440K samples per second over each channel. Internal clocking permits sampling rates from 440K samples per second down to 244 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 16AO20 to sustain continuous throughput to the cable interface independent of the PCI bus interface. The 16AO20 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 16AO20. The specifications provide the information necessary for an in depth understanding of the specialized features implemented on this board.

- The applicable *16AO20 User Manual* from General Standards Corporation.
- The *PCI9080 PCI Bus Master Interface Chip* data handbook from PLX Technology, Inc.

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

1.7. Licensing

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

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
6.0.7	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

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/16ao20` file will be "no".

2.2. The `/proc/` File System

While the driver is running, the text file `/proc/16ao20` can be read to obtain information about the driver. Each file entry includes an entry name followed immediately by a colon, a space character, and the entry value. Below is an example of what appears in the file, followed by descriptions of each entry.

```
version: 3.5.104.47
32-bit support: yes
boards: 1
models: 16AO20
```

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 board models identified by the driver. One model will be listed for each board identified in the system. For this driver the only model numbers listed will be 16AO20.

2.3. File List

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

File	Description
<code>16ao20.linux.tar.gz</code>	This archive contains the driver, the API Library and all related files.
<code>16ao20_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	Content
<code>16ao20/</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 16AO20 API Library (section 4, page 17).
<code>.../docsrc/</code>	This directory contains the code samples from this document (section 6, page 47).
<code>.../driver/</code>	This directory contains the driver and its sources (section 5, page 43).
<code>.../include/</code>	This directory contains the include files for the various libraries.
<code>.../lib/</code>	This directory contains all of the libraries built from the driver archive.

.../samples/	This directory contains the sample applications (section 9, page 51).
.../utils/	This directory contains utility sources used by the sample applications (section 7, page 48).

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 `16ao20.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 `16ao20` in the current directory, and then copies all of the archive's files into this new directory.

```
tar -xzvf 16ao20.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 on page 46.
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 16ao20.linux.tar.gz 16ao20
```

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

```
rm -f /dev/16ao20.*
```

5. If the automated startup procedure was adopted (section 5.3.2, page 44), then edit the system startup script `rc.local` and remove the line that invokes the 16AO20'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 (`.../16ao20/`).
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/lib16ao20_api.so
Defined and Not Empty	==== Linking: ../lib/lib16ao20_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/16ao20_utils.a
Defined and Not Empty	==== Linking: ../lib/16ao20_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 16AO20 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 16AO20 driver archive. 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 16AO20 specific header files. Therefore, sources may include only this one 16AO20 header and make files may reference only this one 16AO20 include directory.

Description	File	Location
Header File	16ao20_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 16AO20 driver archive. 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 pertinent 16AO20 specific static libraries. Therefore, make files may reference only this one 16AO20 static library and only this one 16AO20 library directory.

Description	File	Location
Static Library	16ao20_main.a	.../lib/
	16ao20_multi.a	

NOTE: For applications using the 16AO20 and no other GSC devices, link the 16ao20_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 16AO20 API Library is implemented as a shared library and is thus not linked with the 16AO20 Main Library. The API Library must be linked with applications by adding the argument `-l16ao20_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 rebuilt 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. Rebuild the main library by issuing the below command.

```
make
```

3.2.2. System Libraries

In addition to linking the static library named above, 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

4. API Library

The 16AO20 API Library is the software interface between user applications and the 16AO20 device driver. The interface is accessed by including the header file `16ao20_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.

File	Description
<code>api/*.c</code>	These are library source files.
<code>api/*.h</code>	These are library header files.
<code>api/makefile</code>	This is the library make file.
<code>api/makefile.dep</code>	This is an automatically generated make dependency file.
<code>include/16ao20_api.h</code>	This is the library interface header file.
<code>lib/lib16ao20_api.so</code>	This is the API Library shared library file. *

* The shared 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 API Library shared library is copied to `/usr/lib/`. Therefore, these 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.

```
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. At link time include the below listed linker argument on the linker command line. At link time and at run time the library is found in the directory `/usr/lib/`. (The shared library file is automatically copied to `/usr/lib/` when the library is built.)

Description	File	Location	Linker Argument
Header File	<code>16ao20_api.h</code>	<code>.../include/</code>	
Shared Library	<code>lib16ao20_api.so</code>	<code>.../lib/</code>	
		<code>/usr/lib/</code>	<code>-l16ao20_api</code>

4.4. Macros

The API Library and driver interfaces include the following macros, which are defined in `16ao20.h`.

4.4.1. IOCTL Services

The IOCTL macros are documented in section 4.7 beginning on page 24.

4.4.2. Registers

The following gives the complete set of 16AO20 registers.

4.4.2.1. GSC Registers

The following table gives the complete set of GSC specific 16AO20 registers. For detailed definitions of these registers refer to the relevant 16AO20 User Manual. Please note that the set of registers supported by any given board may vary according to model and firmware version. For the set of supported registers and detailed definitions of these registers please refer to the appropriate *16AO20 User Manual*.

Macro	Description
<code>AO20_GSC_ACR</code>	Adjustable Clock Register
<code>AO20_GSC_AVR</code>	Autocal Values Register
<code>AO20_GSC_BCR</code>	Board Control Register
<code>AO20_GSC_BOR</code>	Buffer Operations Register
<code>AO20_GSC_CSR</code>	Channel Selection Register
<code>AO20_GSC_FOR</code>	Firmware Options Register
<code>AO20_GSC_ODBR</code>	Output Data Buffer Register
<code>AO20_GSC_SRR</code>	Sample Rate Register

4.4.2.2. PCI Configuration Registers

Access to the PCI registers is seldom required so these registers are not listed here. For the complete list of the PCI register identifiers refer to the driver header file `gsc_pci9080.h`, which is automatically included via `16ao20_api.h`.

4.4.2.3. PLX PCI9080 Feature Set Registers

Access to the PLX registers is seldom required so these registers are not listed here. For the complete list of the PLX register identifiers refer to the driver header file `gsc_pci9080.h`, which is automatically included via `16ao20_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.

4.6. Functions

The interface includes the following functions. The return values reflect the completion status of the requested operation. A value of zero indicates success. A negative value indicates that the request could not be completed successfully. The specific value returned is the negative of the corresponding error status value taken from `errno.h`. I/O services return positive values to indicate the number of bytes successfully transferred.

4.6.1. ao20_close()

This function is the entry point to close a connection to an open 16AO20 board. The board is put in an initialized state before this call returns.

Prototype

```
int ao20_close(int fd);
```

Argument	Description
fd	This is the file descriptor of the device to be closed.

Return Value	Description
0	The operation succeeded.
< 0	An error occurred. This is the negative of <code>errno</code> from <code>errno.h</code> .

Example

```
#include <stdio.h>

#include "16ao20_dsl.h"

int ao20_close_dsl(int fd)
{
    int errs;
    int ret;

    ret = ao20_close(fd);

    if (ret)
        printf("ERROR: ao20_close() returned %d\n", ret);

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

4.6.2. ao20_init()

This function is the entry point to initializing the 16AO20 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 ao20_init(void);
```

Return Value	Description
0	The operation succeeded.
< 0	An error occurred. This is the negative of <code>errno</code> from <code>errno.h</code> .

Example

```
#include <stdio.h>
```

```
#include "16ao20_dsl.h"

int ao20_init_dsl(void)
{
    int errs;
    int ret;

    ret = ao20_init();

    if (ret)
        printf("ERROR: ao20_init() returned %d\n", ret);

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

4.6.3. ao20_ioctl()

This function is the entry point to performing setup and control operations on a 16AO20 board. 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 beginning on page 24.

Prototype

```
int ao20_ioctl(int fd, int request, void* arg);
```

Argument	Description
<code>fd</code>	This is the file descriptor of the device to access.
<code>request</code>	This specifies the desired operation to be performed.
<code>arg</code>	This is a request specific argument. Refer to the IOCTL services for additional information (section 4.7, page 24).

Return Value	Description
0	The operation succeeded.
< 0	An error occurred. This is the negative of <code>errno</code> from <code>errno.h</code> .

Example

```
#include <stdio.h>

#include "16ao20_dsl.h"

int ao20_ioctl_dsl(int fd, int request, void *arg)
{
    int errs;
    int ret;

    ret = ao20_ioctl(fd, request, arg);

    if (ret)
        printf("ERROR: ao20_ioctl() returned %d\n", ret);
}
```

```

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

```

4.6.4. ao20_open()

This function is the entry point to open a connection to a 16AO20 board. The device is initialized before the function returns.

Prototype

```
int ao20_open(int device, int share, int* fd);
```

Argument	Description						
device	This is the zero-based index of the 16AO20 to access. *						
share	Open the device in Shared Access Mode? If non-zero the device is opened in Shared Access Mode (see below). If zero the device is opened in Exclusive Access Mode (see below).						
fd	The device handle is returned here. The pointer cannot be NULL. Values returned are as follows. <table border="1" data-bbox="451 835 1268 930"> <thead> <tr> <th>Value</th> <th>Description</th> </tr> </thead> <tbody> <tr> <td>>= 0</td> <td>This is the handle to use to access the device in subsequent calls.</td> </tr> <tr> <td>-1</td> <td>There was an error. The device is not accessible.</td> </tr> </tbody> </table>	Value	Description	>= 0	This is the handle to use to access the device in subsequent calls.	-1	There was an error. The device is not accessible.
Value	Description						
>= 0	This is the handle to use to access the device in subsequent calls.						
-1	There was an error. The device is not accessible.						

* If the index value is -1, then the API Library accesses /proc/16ao20.

Return Value	Description
0	The operation succeeded.
< 0	An error occurred. This is the negative of errno from errno.h.

Example

```

#include <stdio.h>

#include "16ao20_dsl.h"

int ao20_open_dsl(int device, int share, int* fd)
{
    int errs;
    int ret;

    ret = ao20_open(device, share, fd);

    if (ret)
        printf("ERROR: ao20_open() returned %d\n", ret);

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

```

4.6.4.1. Access Modes

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. ao20_read()

This function is the entry point to reading data from an open 16AO20. This function should only be called after a successful open of the respective device. The function reads up to `bytes` bytes from the board. The return value is the number of bytes actually read.

NOTE: When performing an open on device index `-1`, the API Library accesses the `/proc/16ao20` text file. This read service then reads from that file. Refer to section 2.2, page 11.

NOTE: The read service has no functionality for reading from 16AO20 devices. Attempts to read from 16AO20 devices will return an error.

Prototype

```
int ao20_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 will be put here.
<code>bytes</code>	This is the desired number of bytes to read.

Return Value	Description
<code>0</code> to <code>bytes</code>	The operation succeeded. A value less than <code>bytes</code> indicates that the request timed out.
<code>< 0</code>	An error occurred. This is the negative of <code>errno</code> from <code>errno.h</code> .

Example

```
#include <stdio.h>

#include "16ao20_dsl.h"

int ao20_read_dsl(int fd, void* dst, size_t bytes, size_t* qty)
{
    int errs;
    int ret;

    ret = ao20_read(fd, dst, bytes);

    if (ret < 0)
        printf("ERROR: ao20_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. ao20_write()

This function is the entry point to writing data to an open 16AO20. 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: Applications may experience improved responsiveness with read requests by coordinating the Buffer Size setting with the number of samples to write. Refer to the AO20_IOCTL_BUFFER_SIZE service of section 4.7.7 on page 26.

NOTE: When performing an open on device index -1, the API Library accesses the `/proc/16ao20` text file. In this instance, all write requests will fail.

Prototype

```
int ao20_write(int fd, const void *src, size_t bytes);
```

Argument	Description
<code>fd</code>	This is the file descriptor of use for access.
<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. This is the negative of <code>errno</code> from <code>errno.h</code> .

Example

```

#include <stdio.h>

#include "16ao20_dsl.h"

int ao20_write_dsl(int fd, const void* src, size_t bytes, size_t*
qty)
{
    int errs;
    int ret;

    ret = ao20_write(fd, src, bytes);

    if (ret < 0)
        printf("ERROR: ao20_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 16AO20 API Library and device driver implement the following IOCTL services. Each service is described along with the applicable `ao20_ioctl()` function arguments.

4.7.1. AO20_IOCTL_AUTO_CAL_STS

This service retrieves the auto-calibration completion status.

Usage

Argument	Description
request	AO20_IOCTL_AUTO_CAL_STS
arg	s32*

The value returned will be one of the following.

Value	Description
AO20_AUTO_CAL_STS_ACTIVE	Auto-calibration is in progress.
AO20_AUTO_CAL_STS_FAIL	Auto-calibration failed.
AO20_AUTO_CAL_STS_PASS	Auto-calibration passed.

4.7.2. AO20_IOCTL_AUTO_CALIBRATE

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

NOTE: Do not access the board while an auto-calibration cycle is in progress. Doing so may produce indeterminate results, and may lockup the board.

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

Usage

Argument	Description
request	AO20_IOCTL_AUTO_CALIBRATE
arg	Not used.

4.7.3. AO20_IOCTL_BUFFER_CLEAR

This service immediately clears the current content from the output buffer. It also clears the board's buffer overrun and frame overrun status. This service does not halt data output.

Usage

Argument	Description
request	AO20_IOCTL_BUFFER_CLEAR
arg	Not used.

4.7.4. AO20_IOCTL_BUFFER_MODE

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

Usage

Argument	Description
request	AO20_IOCTL_BUFFER_MODE
arg	s32*

Valid argument values are as follows.

Value	Description
-1	Retrieve the current setting.
AO20_BUFFER_MODE_CIRC	Buffer data is recycled when it exits the buffer.
AO20_BUFFER_MODE_OPEN	The buffer data is not recycled when it exits the buffer.

4.7.5. AO20_IOCTL_BUFFER_OVERFLOW_DATA

This service operates on the Buffer Overflow status.

Usage

Argument	Description
request	AO20_IOCTL_BUFFER_OVERFLOW_DATA
arg	s32*

Valid argument values are as follows.

Value	Description
AO20_BUFFER_OVERFLOW_DATA_CHK	Report if an overflow has occurred.
AO20_BUFFER_OVERFLOW_DATA_CLR	Clear the overflow status.

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

Value	Description
AO20_BUFFER_OVERFLOW_DATA_NO	An overflow did not occur.
AO20_BUFFER_OVERFLOW_DATA_YES	An overflow did occur.

4.7.6. AO20_IOCTL_BUFFER_OVERFLOW_FRAME

This service operates on the Frame Overflow status.

Usage

Argument	Description
request	AO20_IOCTL_BUFFER_OVERFLOW_FRAME
arg	s32*

Valid argument values are as follows.

Value	Description
AO20_BUFFER_OVERFLOW_FRAME_CHK	Report if an overflow has occurred.
AO20_BUFFER_OVERFLOW_FRAME_CLR	Clear the overflow status.

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

Value	Description
AO20_BUFFER_OVERFLOW_NO	An overflow did not occur.
AO20_BUFFER_OVERFLOW_YES	An overflow did occur.

4.7.7. AO20_IOCTL_BUFFER_SIZE

This service configures the active size of the output buffer.

Usage

Argument	Description
request	AO20_IOCTL_BUFFER_SIZE
arg	s32*

Valid argument values are as follows.

Value	Description
-1	Retrieve the current setting.
AO20_BUFFER_SIZE_8	Set the buffer's active size to 8 samples.
AO20_BUFFER_SIZE_16	Set the buffer's active size to 16 samples.
AO20_BUFFER_SIZE_32	Set the buffer's active size to 32 samples.
AO20_BUFFER_SIZE_64	Set the buffer's active size to 64 samples.
AO20_BUFFER_SIZE_128	Set the buffer's active size to 128 samples.
AO20_BUFFER_SIZE_256	Set the buffer's active size to 256 samples.
AO20_BUFFER_SIZE_512	Set the buffer's active size to 512 samples.
AO20_BUFFER_SIZE_1K	Set the buffer's active size to 1K samples (1,024).
AO20_BUFFER_SIZE_2K	Set the buffer's active size to 2K samples (2,048).
AO20_BUFFER_SIZE_4K	Set the buffer's active size to 4K samples (4,096).
AO20_BUFFER_SIZE_8K	Set the buffer's active size to 8K samples (8,192).
AO20_BUFFER_SIZE_16K	Set the buffer's active size to 16K samples (16,384).
AO20_BUFFER_SIZE_32K	Set the buffer's active size to 32K samples (32,768).
AO20_BUFFER_SIZE_64K	Set the buffer's active size to 64K samples (65,536).
AO20_BUFFER_SIZE_128K	Set the buffer's active size to 128K samples (131,072).
AO20_BUFFER_SIZE_256K	Set the buffer's active size to 256K samples (262,144).

4.7.8. AO20_IOCTL_BUFFER_STATUS

This service reports the relative fill level of the active buffer. The buffer's active size is set with the AO20_IOCTL_BUFFER_SIZE service (section 4.7.7, page 26).

Usage

Argument	Description
request	AO20_IOCTL_BUFFER_STATUS
arg	s32*

The service returns one of the following values.

Value	Description
AO20_BUFFER_STATUS_EMPTY	The buffer is empty.
AO20_BUFFER_STATUS_1Q_FULL	The buffer is less than ¼ full.
AO20_BUFFER_STATUS_MEDIUM	The buffer is from ¼ to ¾ full.

AO20_BUFFER_STATUS_3Q_FULL	The buffer is ¾ full or more.
AO20_BUFFER_STATUS_FULL	The buffer is full.

4.7.9. AO20_IOCTL_BURST_ENABLE

This service enables or disables output bursting.

Usage

Argument	Description
request	AO20_IOCTL_BURST_ENABLE
arg	s32*

Valid argument values are as follows.

Value	Description
-1	Retrieve the current state.
AO20_BURST_ENABLE_NO	This refers to output bursting being disabled.
AO20_BURST_ENABLE_YES	This refers to output bursting being enabled.

4.7.10. AO20_IOCTL_BURST_READY

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

Usage

Argument	Description
request	AO20_IOCTL_BURST_READY
arg	s32*

The service returns one of the following values.

Value	Description
AO20_BURST_READY_NO	The board is not ready for burst initiation.
AO20_BURST_READY_YES	The board is ready for burst initiation.

4.7.11. AO20_IOCTL_BURST_TRIG_SRC

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

Usage

Argument	Description
request	AO20_IOCTL_BURST_TRIG_SRC
arg	s32*

Valid argument values are as follows.

Value	Description
-1	Retrieve the current setting.
AO20_BURST_TRIG_SRC_EXT	Utilize external burst triggering.
AO20_BURST_TRIG_SRC_SW	Utilize software burst triggering.

4.7.12. AO20_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 (though not the infinite option). (See AO20_IOCTL_TX_IO_TIMEOUT, section 4.7.40, page 38.)

Usage

Argument	Description
request	AO20_IOCTL_BURST_TRIGGER
arg	Not used.

4.7.13. AO20_IOCTL_CHANNEL_SEL

This service enables or disables channels for outputting data.

Usage

Argument	Description
request	AO20_IOCTL_CHANNEL_SEL
arg	s32*

Valid argument values are any valid set of bits for the set of supported channels, and -1. If a bit is set, then the corresponding channel is enabled. A zero bit disables the channel. The lowest significant bit corresponds to channel zero. The value -1 is used to retrieve the current setting.

4.7.14. AO20_IOCTL_CLOCK_ENABLE

This service enables and disables clocking of output data.

Usage

Argument	Description
request	AO20_IOCTL_CLOCK_ENABLE
arg	s32*

Valid argument values are as follows.

Value	Description
-1	Retrieve the current setting.
AO20_CLOCK_ENABLE_NO	This disables the output sample clock.
AO20_CLOCK_ENABLE_YES	This enables the output sample clock.

4.7.15. AO20_IOCTL_CLOCK_OSC_SRC

This service selects the rate generator's clock source.

Usage

Argument	Description
request	AO20_IOCTL_CLOCK_OSC_SRC
arg	s32*

Valid argument values are as follows.

Value	Description
-1	Retrieve the current setting.
AO20_CLOCK_OSC_SRC_ALT	This selects the alternate source, which is generally derived from the 16MHz source. *
AO20_CLOCK_OSC_SRC_PRI	This selects the primary source, which generally refers to the 30MHz oscillator. *

* Refer to the hardware reference manual for additional information.

4.7.16. AO20_IOCTL_CLOCK_READY

This service reports the board's readiness to accept a software or external clock.

Usage

Argument	Description
request	AO20_IOCTL_CLOCK_READY
arg	s32*

The service returns one of the following values.

Value	Description
AO20_CLOCK_READY_NO	The board is not ready for a clock.
AO20_CLOCK_READY_YES	The board is ready for a clock.

4.7.17. AO20_IOCTL_CLOCK_SRC

This service selects the source for the output sample clock.

Usage

Argument	Description
request	AO20_IOCTL_CLOCK_SRC
arg	s32*

Valid argument values are as follows.

Value	Description
-1	Retrieve the current setting.
AO20_CLOCK_SRC_EXT_SW	This selects the external/software option as the clock source.
AO20_CLOCK_SRC_INT	This selects the internal rate generator as the source.

4.7.18. AO20_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	AO20_IOCTL_CLOCK_SW
arg	Not used.

4.7.19. AO20_IOCTL_DATA_FORMAT

This service sets the data encoding format.

Usage

Argument	Description
request	AO20_IOCTL_DATA_FORMAT
arg	s32*

Valid argument values are as follows.

Value	Description
-1	Retrieve the current setting.
AO20_DATA_FORMAT_2S_COMP	Select the Twos Compliment data format.
AO20_DATA_FORMAT_OFF_BIN	Select the Offset Binary encoding format.

4.7.20. AO20_IOCTL_EXT_CLK_IN_POL

This service configures the polarity of the External Clock Input signal at the cable interface.

Usage

Argument	Description
request	AO20_IOCTL_EXT_CLK_IN_POL
arg	s32*

Valid argument values are as follows.

Value	Description
-1	Retrieve the current setting.
AO20_EXT_POL_INVERT	This selects the inverted state.
AO20_EXT_POL_NORMAL	This selects the normal, default state.

4.7.21. AO20_IOCTL_EXT_TRIG_IN_POL

This service configures the polarity of the External Trigger Input signal at the cable interface.

Usage

Argument	Description
request	AO20_IOCTL_EXT_TRIG_IN_POL
arg	s32*

Valid argument values are as follows.

Value	Description
-1	Retrieve the current setting.
AO20_EXT_POL_INVERT	This selects the inverted state.
AO20_EXT_POL_NORMAL	This selects the normal, default state.

4.7.22. AO20_IOCTL_EXT_TRIG_OUT_POL

This service configures the polarity of the External Trigger Output signal at the cable interface.

Usage

Argument	Description
request	AO20_IOCTL_EXT_TRIG_IN_POL
arg	s32*

Valid argument values are as follows.

Value	Description
-1	Retrieve the current setting.
AO20_EXT_POL_INVERT	This selects the inverted state.
AO20_EXT_POL_NORMAL	This selects the normal, default state.

4.7.23. AO20_IOCTL_GROUND_SENSE

This service configures the board's ground sense logic.

Usage

Argument	Description
request	AO20_IOCTL_GROUND_SENSE
arg	s32*

Valid argument values are as follows.

Value	Description
-1	Retrieve the current setting.
AO20_GROUND_SENSE_DISABLE	This disables remote ground sensing.
AO20_GROUND_SENSE_REMOTE	This selects remote ground sensing.

4.7.24. AO20_IOCTL_INITIALIZE

This service returns all driver interface settings for the board to the state they were in when the board was first opened. This includes both hardware-based settings and software-based settings.

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	AO20_IOCTL_INITIALIZE
arg	Not used.

4.7.25. AO20_IOCTL_IRQ_SEL

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

Usage

Argument	Description
request	AO20_IOCTL_IRQ_SEL
arg	s32*

Valid argument values are as follows.

Value	Description
-1	Retrieve the current setting.
AO20_IRQ_AUTO_CAL_DONE	This refers to the completion of auto-calibration.
AO20_IRQ_BUF_1Q_FULL	This refers to the buffer becoming less than ¼ full.
AO20_IRQ_BUF_3Q_FULL	This refers to the buffer becoming more than 3/4 full.
AO20_IRQ_BUF_EMPTY	This refers to the buffer becoming empty.
AO20_IRQ_BURST_TRIG_READY	This refers to the board becoming ready for a burst trigger.
AO20_IRQ_INIT_DONE	This refers to the completion of initialization.
AO20_IRQ_LOAD_READY	This refers to a circular buffer becoming ready to receive data.
AO20_IRQ_LOAD_READY_END	This refers to a circular buffer becoming not ready to receive data.

4.7.26. AO20_IOCTL_LOAD_READY

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

Usage

Argument	Description
request	AO20_IOCTL_LOAD_READY
arg	s32*

Valid values returned by the service are as follows.

Value	Description
AO20_LOAD_READY_NO	The buffer is not ready to receive additional data.
AO20_LOAD_READY_YES	The buffer is ready to receive additional data.

4.7.27. AO20_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	AO20_IOCTL_LOAD_REQUEST
arg	Not used.

4.7.28. AO20_IOCTL_NCLK

This service sets the adjustable clock's NCLK adjustment value.

Usage

Argument	Description
request	AO20_IOCTL_NCLK
arg	s32*

Valid argument values are in the range from 0 to 0x1FF, and -1. The value -1 is used to retrieve the current setting.

4.7.29. AO20_IOCTL_NRATE

This service sets the rate divider's NRATE adjustment value.

Usage

Argument	Description
request	AO20_IOCTL_NRATE
arg	s32*

Valid argument values are in the range from 68 to 0xFFFF, and -1. The value -1 is used to retrieve the current setting.

4.7.30. AO20_IOCTL_OUTPUT_MODE

This service configures the buffer's data output mode.

Usage

Argument	Description
request	AO20_IOCTL_OUTPUT_MODE
arg	s32*

Valid argument values are as follows.

Value	Description
-1	Retrieve the current setting.
AO20_BURST_OUTPUT_MODE_SEQ	Channel data is output to one channel at a time, sequentially. *
AO20_BURST_OUTPUT_MODE_SIM	Channel data is output to all channels simultaneously.

* Refer to the hardware reference manual to see how this affects the sample rate.

4.7.31. AO20_IOCTL_QUERY

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

Usage

Argument	Description
request	AO20_IOCTL_QUERY
arg	s32*

Valid argument values are as follows.

Value	Description
AO20_QUERY_AUTO_CAL_MS	This returns the maximum duration of the Auto Calibration cycle in milliseconds.
AO20_QUERY_CHANNEL_MASK	This is the valid mask for all selectable channels and is based on the number of channels the board supports.
AO20_QUERY_CHANNEL_MAX	This returns the maximum number of output channels supported by all boards of the same model as the board accessed.
AO20_QUERY_CHANNEL_QTY	This returns the actual number of output channels on the current board.

AO20_QUERY_COUNT	This returns the number of query options supported by the IOCTL service.
AO20_QUERY_DEVICE_TYPE	This returns the identifier value for the board's type. The value is a member of the <code>gsc_dev_type_t</code> enumeration, which is defined in <code>gsc_common.h</code> .
AO20_QUERY_FIFO_SIZE	This returns the size of the output buffer in samples.
AO20_QUERY_FILTER	This returns the identifier for the installed filter option. Refer to the table below for the returned option values.
AO20_QUERY_FREF_DEFAULT	This gives the default FREF value in hertz.
AO20_QUERY_FSAMP_MAX	This gives the maximum sample rate in S/S.
AO20_QUERY_FSAMP_MIN	This gives the minimum sample rate in S/S.
AO20_QUERY_INIT_MS	This returns the duration of a board initialization in milliseconds.
AO20_QUERY_LAST	This is included for reference only and should not be used by applications. Applications should use the COUNT option instead.
AO20_QUERY_NCLK_MASK	This returns the mask for the adjustable clock's NCLK value.
AO20_QUERY_NCLK_MAX	This returns the maximum supported NCLK value.
AO20_QUERY_NCLK_MIN	This returns the minimum supported NCLK value.
AO20_QUERY_NRATE_MASK	This returns the mask for the board's NRATE field.
AO20_QUERY_NRATE_MAX	This returns the maximum supported NRATE value.
AO20_QUERY_NRATE_MIN	This returns the minimum supported NRATE value.
AO20_QUERY_RANGE	This returns the identifier for the installed voltage range option. Refer to the table below for the returned option values.
AO20_QUERY_EXT_CLK_IN_POL	This returns an indication of the External Clock Input Polarity option is supported (section 4.7.20, page 30).
AO20_QUERY_EXT_TRIG_IN_POL	This returns an indication of the External Trigger Input Polarity option is supported (section 4.7.21, page 30).
AO20_QUERY_EXT_TRIG_OUT_POL	This returns an indication of the External Trigger Output Polarity option is supported (section 4.7.22, page 30).

Valid return values for the AO20_QUERY_FILTER option are as follows.

Value	Description
AO20_FILTER_NONE	No filter is installed.
AO20_FILTER_10KHZ	A 10KHz filter is installed.
AO20_FILTER_100KHZ	A 100KHz filter is installed.

Valid return values for the AO20_QUERY_RANGE option are as follows.

Value	Description
AO20_RANGE_2_5	The board is hardwired for the voltage range of $\pm 2.5V$.
AO20_RANGE_5	The board is hardwired for the voltage range of $\pm 5V$.
AO20_RANGE_10	The board is hardwired for the voltage range of $\pm 10V$.

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

Value	Description
AO20_IOCTL_QUERY_ERROR	Either there was a processing error or the query option is unrecognized.

4.7.32. AO20_IOCTL_REG_MOD

This service performs a read-modify-write of a 16AO20 register. This includes only the GSC firmware registers. The PCI and PLX Feature Set Registers are read-only. Refer to `16ao20.h` for a complete list of the GSC firmware registers.

Usage

Argument	Description
request	AO20_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. AO20_IOCTL_REG_READ

This service reads the value of a 16AO20 register. This includes the PCI registers, the PLX Feature Set Registers and the GSC firmware registers. Refer to `16ao20.h` and `gsc_pci9080.h` for the complete list of accessible registers.

Usage

Argument	Description
request	AO20_IOCTL_REG_READ
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 is the value read from the specified register.
mask	This is ignored for read request.

4.7.34. AO20_IOCTL_REG_WRITE

This service writes a value to a 16AO20 register. This includes only the GSC firmware registers. The PCI and PLX Feature Set Registers are read-only. Refer to `16ao20.h` for a complete list of the GSC firmware registers.

Usage

Argument	Description
request	AO20_IOCTL_REG_WRITE
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 is the value to write to the specified register.
mask	This is ignored for write request.

4.7.35. AO20_IOCTL_SAMPLE_MODE

This service controls how and when data is output to the cable interface.

Usage

Argument	Description
request	AO20_IOCTL_SAMPLE_MODE
arg	s32*

Valid argument values are as follows.

Value	Description
-1	Retrieve the current setting.
AO20_SAMPLE_MODE_BURST	This selects triggered burst operation.
AO20_SAMPLE_MODE_CONT	This selects continuous output operation.

4.7.36. AO20_IOCTL_TX_IO_ABORT

This service aborts an ongoing `ao20_write()` request.

Usage

Argument	Description
request	AO20_IOCTL_TX_IO_ABORT
arg	s32*

The results are reported as one of the following values.

Value	Description
AO20_IO_ABORT_NO	An ao20 write() request was not aborted as none were ongoing.
AO20_IO_ABORT_YES	An ongoing ao20 write() request was aborted.

4.7.37. AO20_IOCTL_TX_IO_MODE

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

NOTE: Applications may experience improved responsiveness with write requests by coordinating the Active Buffer Size with the number of samples in the write request. Refer to the AO20_IOCTL_BUFFER_SIZE service of section 4.7.7 on page 26.

Usage

Argument	Description
request	AO20_IOCTL_TX_IO_MODE
arg	s32*

Valid argument values are as follows.

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

4.7.38. AO20_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	AO20_IOCTL_TX_IO_OVER_DATA
arg	s32*

Valid argument values are as follows.

Value	Description
-1	Retrieve the current setting.
AO20_TX_IO_OVER_DATA_CHECK	Perform the check. This is the default.
AO20_TX_IO_OVER_DATA_IGNORE	Do not perform the check.

4.7.39. AO20_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	AO20_IOCTL_TX_IO_OVER_FRAME
arg	s32*

Valid argument values are as follows.

Value	Description
-1	Retrieve the current setting.
AO20_TX_IO_OVER_FRAME_CHECK	Perform the check. This is the default.
AO20_TX_IO_OVER_FRAME_IGNORE	Do not perform the check.

4.7.40. AO20_IOCTL_TX_IO_TIMEOUT

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

Usage

Argument	Description
request	AO20_IOCTL_TX_IO_TIMEOUT
arg	s32*

Valid argument values are in the range from zero to 3600, -1, and AO20_IOCTL_TIMEOUT_INFINITE. A value of zero tells the driver not to sleep in order to wait for more space, and should only be used with PIO mode reads. A value of -1 is used to retrieve the current setting. If the option AO20_IOCTL_TIMEOUT_INFINITE is used, then the driver will wait indefinitely rather than timing out. The default is 10 seconds.

4.7.41. AO20_IOCTL_WAIT_CANCEL

This service resumes all threads blocked via AO20_IOCTL_WAIT_EVENT IOCTL calls (section 4.7.42, 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	AO20_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 4.7.42.2 on page 40.

gsc	This specifies the set of AO20_WAIT_GSC_* events whose wait requests are to be cancelled. Refer to section 4.7.42.3 on page 40.
alt	This is unused by the 16AO20 driver and should be zero.
io	This specifies the set of AO20_WAIT_IO_* events whose wait requests are to be cancelled. Refer to section 4.7.42.4 on page 40.
timeout_ms	This is unused by wait cancel operations.
count	Upon return this indicates the number of waits that were cancelled.

4.7.42. AO20_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	AO20_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 4.7.42.1 on page 40.
main	This specifies any number of GSC_WAIT_MAIN_* events that the thread is to wait for. Refer to section 4.7.42.2 on page 40.
gsc	This specifies any number of AO20_WAIT_GSC_* events that the thread is to wait for. Refer to section 4.7.42.3 on page 40.
alt	This is unused by the 16AO20 driver and must be zero.
io	This specifies any number of AO20_WAIT_IO_* events that the thread is to wait for. Refer to section 4.7.42.4 on 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.

4.7.42.1. `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.

4.7.42.2. `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 16AO20 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 16AO20.
<code>GSC_WAIT_MAIN_PCI</code>	This refers to any interrupt generated by the 16AO20.
<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.

4.7.42.3. `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 `AO20_IOCTL_IRQ_SEL` (section 4.7.25, page 31).

Value	Description
<code>AO20_WAIT_GSC_AUTO_CAL_DONE</code>	This refers to the completion of auto-calibration.
<code>AO20_WAIT_GSC_BUF_1Q_FULL</code>	This refers to the buffer becoming less than ¼ full.
<code>AO20_WAIT_GSC_BUF_3Q_FULL</code>	This refers to the buffer becoming more than ¾ full.
<code>AO20_WAIT_GSC_BUF_EMPTY</code>	This refers to the buffer becoming empty.
<code>AO20_WAIT_GSC_BURST_TRIG_READY</code>	This refers to the board becoming ready for a burst trigger.
<code>AO20_WAIT_GSC_INIT_DONE</code>	This refers to the completion of initialization.
<code>AO20_WAIT_GSC_LOAD_READY</code>	This refers to a circular buffer becoming ready to receive data.
<code>AO20_WAIT_GSC_LOAD_READY_END</code>	This refers to a circular buffer becoming not ready to receive data.

4.7.42.4. `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>AO20_WAIT_IO_TX_ABORT</code>	This refers to write requests which have been aborted.
<code>AO20_WAIT_IO_TX_DONE</code>	This refers to write requests which have been satisfied.
<code>AO20_WAIT_IO_TX_ERROR</code>	This refers to write requests which end due to an error.
<code>AO20_WAIT_IO_TX_TIMEOUT</code>	This refers to write requests which end due to the timeout period lapse.

4.7.43. AO20_IOCTL_WAIT_STATUS

This service count all threads blocked via the AO20_IOCTL_WAIT_EVENT IOCTL service (section 4.7.42, 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	AO20_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 4.7.42.2 on page 40.
gsc	This specifies the set of AO20_WAIT_GSC_* events whose wait requests are to be counted. Refer to section 4.7.42.3 on page 40.
alt	This is unused by the 16AO20 driver and should be zero.
io	This specifies the set of AO20_WAIT_IO_* events whose wait requests are to be counted. Refer to section 4.7.42.4 on 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.

4.7.44. AO20_IOCTL_XCVR_TYPE

This service selects TTL or LVDS signaling on the external digital cable signals.

Usage

Argument	Description
request	AO20_IOCTL_XCVR_TYPE
arg	s32*

Valid argument values are as follows.

Value	Description
-1	Retrieve the current setting.
AO20_XCVR_TYPE_LVDS	Use LVDS signaling.
AO20_XCVR_TYPE_TTL	Use TTL signaling.

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.

File	Description
driver/*.c	The driver source files.
driver/*.h	The driver header files.
driver/16ao20.h	This is the driver interface header file.
driver/Makefile	This is the driver make file.
driver/start	Shell script to install the driver executable and device nodes.
driver/16ao20.ko	This is the driver executable (kernel 2.6 and later).
driver/16ao20.o	This is the driver executable (kernel 2.4 and earlier).

5.2. Build

NOTE: Building the driver requires installation of the kernel headers.

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 device driver. 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 correspond 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 rebooted.

NOTE: The 16AO20 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 16ao20 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/16ao20.*
```

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/16ao20/driver/start
```

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

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

5.3.2.1. File `rc.local` Not Present

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

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

5.3.2.2. Default `rc.local` File Permissions

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

5.3.2.3. `systemd` Installations

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

```
systemctl enable rc-local
```

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

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

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

5.3.2.5. SELinux Implications

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

```
yum install setroubleshoot setools
```

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

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

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

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

5.4. Verification

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

1. Verify that the file `/proc/16ao20` 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/16ao20
```

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/16ao20` 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 16ao20
```

2. Verify that the driver module has been unloaded by issuing the below command. The module name `16ao20` 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.

File	Description
docsrc/*.c	These are the C source files.
docsrc/makefile	This is the library make file.
docsrc/makefile.dep	This is an automatically generated make dependency file.
include/16ao20_dsl.h	This is the primary utility header file.
lib/16ao20_dsl.a	This is the statically linkable library file.

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 below listed header file in each source file using a component of the library interface. At link time include the below listed library file with the objects being linked with the application.

Description	File	Location
Header File	16ao20_dsl.h	.../include/
Static Link Library	16ao20_dsl.a	.../lib/

7. Utility Source Code

The driver archive includes a body of utility services built into a statically linkable library that is usable with console applications. The primary purpose of the services is both for code reuse in the sample applications and to provide wrappers, mostly visual, around the driver's IOCTL services. The aim of the visual wrappers is to facilitate structured console output for the sample applications. 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.

7.1. Files

The library files are summarized in the table below.

File	Description
utils/*.c	These are utility source files.
utils/gsc_*.c	These are device and OS independent utility source files.
utils/os_*.c	These are OS specific utility source files.
utils/makefile	This is the library make file.
utils/makefile.dep	This is an automatically generated make dependency file.
include/16ao20_utils.h	This is the primary utility header file.
lib/16ao20_utils.a	This is the statically linkable library file.

7.2. 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 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 below listed header file in each source file using a component of the library interface. At link time include the below listed library files with the objects being linked with the application.

Description	File	Location
Header File	16ao20_utils.h	.../include/
Static Link Libraries	16ao20_utils.a gsc_utils.a os_utils.a plx_utils.a	.../lib/

8. Operating Information

This section explains some basic operational procedures for using the 16AO20. 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. Analog Output Configuration

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

Item	Name/File	Location
Function	ao20_config_ao()	Source File
Source File	util_config_ao.c	.../utils/
Header File	16ao20_utils.h	.../include/
Library File	16ao20_utils.a	.../lib/

8.2. Debugging Aids

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

8.2.1. 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/

8.2.2. Detailed Register Dump

Among the utility services provided is a function to generate a detailed listing of the board's registers to the console. When used, the function is typically used to verify the board's configuration. In these cases, the function should be called just prior to the first read or write operation. When intended for sending to GSC tech support, please set the *detail* argument 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	ao20_reg_list()	Source File
Source File	util_reg.c	.../utils/
Header File	16ao20_utils.h	.../include/
Library File	16ao20_utils.a	.../lib/

8.3. Data Transfer Modes

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 will read data from the input buffer register until either the

buffer is empty, or the I/O timeout expires, whichever occurs first. This is generally the least efficient mode, but for very small transfers it is more efficient than DMA.

8.3.2. BMDMA - Block Mode DMA

For DMA transfers, hardware onboard the 16AO20 is used to transfer the data without processor intervention. In this mode the driver checks for available space in the output buffer. When sufficient space is available a DMA transfer is performed. Depending on the size of the write request, the driver may break the request into smaller transfers in order to ensure data integrity. The breakup is based on the size of the request relative to the size of the active buffer. If the active buffer is empty, then the driver will perform a DMA transfer for up to the size of the active buffer. If the active buffer is up to $\frac{1}{4}$ full, then the driver will perform a DMA transfer for up to $\frac{3}{4}$ the size of the active buffer. If the active buffer is from $\frac{1}{4}$ to $\frac{3}{4}$ full, then the driver will perform a DMA transfer for up to the $\frac{1}{4}$ the size of the active buffer. If the active buffer is $\frac{3}{4}$ full or more, then the driver will sleep for one system timer tick and then check again. The process is repeated until the data is exhausted or the I/O timeout expires, whichever occurs first.

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

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

9.5. sbtest - Single Board Test - .../sbtest/

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

9.6. 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
April 5, 2023	Updated to version 3.5.104.47.0. Updated the kernel support table. Various editorial changes. Added section on environment variables. Updated the information for the open and close calls.
February 23, 2022	Updated to version 3.4.96.38.0. Updated the kernel support table. Minor editorial changes. Added a licensing subsection. Added WAIT_EVENT note. Expanded automatic startup information.
May 16, 2019	Updated to version 3.3.85.27.0. Minor editorial changes. Various paragraph rewrites. Numerous table rewrites.
March 7, 2019	Updated to version 3.3.82.26.0. Updated Block Mode DMA macro and associated information. Minor editorial changes.
July 6, 2018	Updated to version 3.2.78.24.0. Updated the inside cover page. Updated the CPU and kernel support section. Minor editorial changes. Document reorganization.
December 1, 2016	Updated to version 3.1.68.18.0. Removed the <code>built</code> field from the <code>/proc</code> file. Updated the kernel support table. Updated the sample applications for the <code>fsamp</code> sample application. 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. Added support for infinite I/O timeouts. Updated the operating information section. Made various miscellaneous updates. Some document reorganization.
September 14, 2015	Updated to version 3.0.60.8.0. Updated the device node name to include a period before the device index. Removed double underscore that prefaced various data types.
March 13, 2015	Updated to version 2.6.57.0. Added the External Signal Polarity options.
February 28, 2014	Updated to version 2.5.52.0. Updated the kernel support table.
January 9, 2014	Updated to version 2.4.51.0. Updated the kernel support table.
November 12, 2013	Updated to version 2.4.49.0.
July 17, 2013	Updated to version 2.4.45.0. Updated the kernel support table.
November 14, 2011	Updated to version 2.3.32.0.
February 7, 2011	Updated to version 2.2.22.0. Various editorial changes. Removed the <code>IRQ_STATE</code> and <code>IRQ_ENABLE</code> IOCTL services. Renamed the <code>IRQ_SEL</code> IOCTL service options to <code>IRQ</code> . Updated the CPU and Kernel Support information. Updated the comments for the Initialize IOCTL service. Changed the spelling of various Auto Calibration related software items.
December 28, 2009	Updated to version 2.1.13.0. Made other minor modifications.
August 23, 2009	Updated to version 2.0.9.0.
May 26, 2009	Updated to version 2.0.7.1.
May 19, 2009	This is the initial release of the 2.x driver.