16AISS8AO4

16-Bit Analog I/O, 8 Input Channels, 4 Output Channels
16-Bit Digital I/O

PMC66-16AISS8AO4

Linux Device Driver
And API Library
User Manual

Manual Revision: March 14, 2024
Driver Release Version 4.5.109.50.0

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

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

Table of Contents

Table of Figures

1. Introduction

1.1. Purpose

The purpose of this document is to describe the interface to the 16AISS8AO4 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 16AISS8AO4 hardware. The API Library and driver interfaces are based on the board's functionality.

1.2. Acronyms

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

Acronyms	Description
ADC	Analog-to-Digital Converter
API	Application Programming Interface
BMDMA	Block Mode DMA
DAC	Digital-to-Analog Converter
DMA	Direct Memory Access
DMDMA	Demand Mode DMA
GSC	General Standards Corporation
PCI	Peripheral Component Interconnect
PIO	Programmed I/O
PMC	PCI Mezzanine Card
PMC66	This is a PMC formfactor device that can operate at up to 66MHz over the PCI bus.

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 16AISS8AO4 installation directory or any of its subdirectories.
16AISS8AO4	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 16AISS8AO4 hardware.
Application	This is a user mode process, which runs in user space with user mode privileges.
Driver	This is the 16AISS8AO4 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 16AISS8AO4 applications. The overall architecture is illustrated in Figure 1 below.

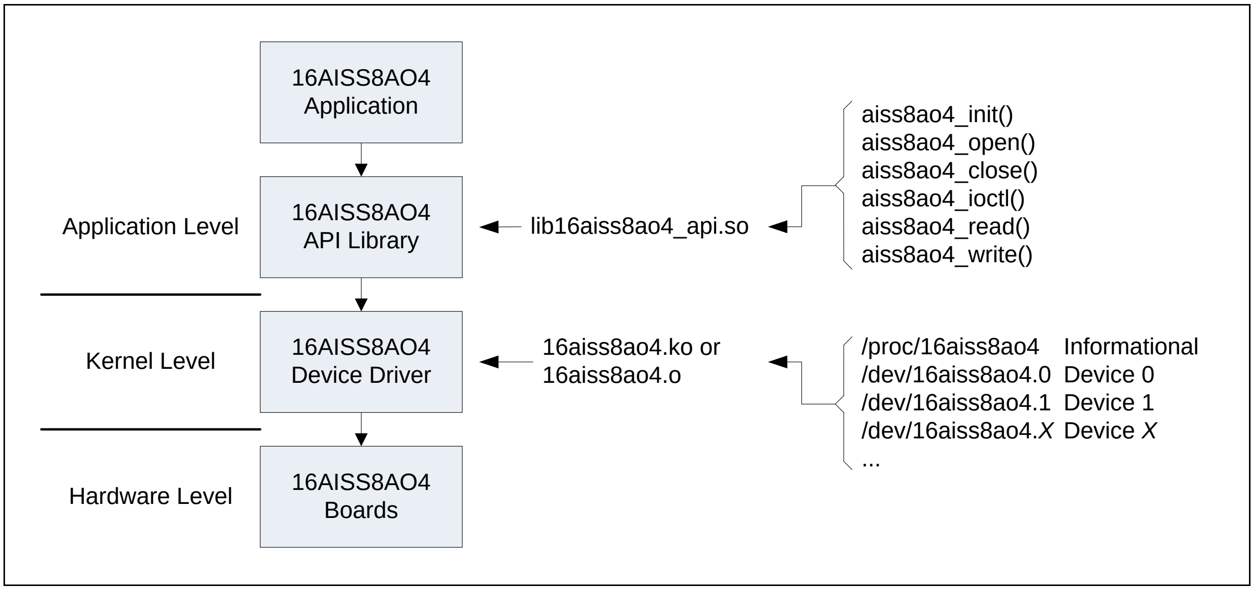

Figure 1 Basic architectural representation.

1.4.2. API Library

The primary means of accessing 16AISS8AO4 boards is via the 16AISS8AO4 API Library. This library forms a layer between the application and the driver. Additional information is given in section 4 (page 18). With the library, applications are able to open and close a device and, while open, perform I/O control, read and write operations.

1.4.3. Device Driver

The device driver is the host software that provides a means of communicating directly with 16AISS8AO4 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 16AISS8AO4 is a high-performance 16-bit analog I/O board. There are eight input channels, four output channels, and two 8-bit digital I/O ports. The host side connection is PCI based and the form factor is according to the model ordered. The board is capable of acquiring data at up to 2M samples per second over each input channel and can output data at up to 1M samples per second over each output channel. Clocking for both the input and the output can come from either an on-board clock or from the cable interface. Onboard storage permits data buffering between the PCI bus and the cable interface of up to 256K samples for input and 256K samples for output. This allows the 16AISS8AO4 to sustain continuous throughput over the cable interface independent of the PCI bus interface. The 16AISS8AO4 also permits multiple boards to be synchronized so that all boards sample data in unison. In addition, the board includes autocalibration capability.

1.6. Reference Material

The following reference material may be of particular benefit in using the 16AISS8AO4. The specifications provide the information necessary for an in depth understanding of the specialized features implemented on this board.

- The applicable *16AISS8AO4 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/16aiss8ao4` file will be "`no`".

2.2. The /proc/ File System

While the driver is running, the text file `/proc/16aiss8ao4` 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: 4.5.109.50
32-bit support: yes
boards: 1
models: 16AISS8AO4
```

Entry	Description
`version`	This gives the driver version number in the form `x.x.x.x`.
`32-bit support`	This reports the driver's support for 32-bit applications. This will be either "`yes`" or "`no`" for 64-bit driver builds and "`yes (native)`" for 32-bit builds.
`boards`	This identifies the total number of 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
`16aiss8ao4.linux.tar.gz`	This archive contains the driver, the API Library and all related files.
`16aiss8ao4_linux_um.pdf`	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
`16aiss8ao4/`	This is the driver root directory. It contains the documentation, the Overall Make Script (section 2.7, page 13) and the below listed subdirectories.
…`/api/`	This directory contains the API Library source files (section 4, page 18).
…`/docsrc/`	This directory contains the source files for the code samples given in this document (section 1, page 57).
…`/driver/`	This directory contains the device driver source files (section 5, page 54).
…`/include/`	This directory contains the header files for the various libraries.

…/lib/	This directory contains all of the libraries built from the installed sources.
…/samples/	This directory contains the sample application subdirectories and all of their corresponding source files (section 9, page 62).
…/utils/	This directory contains the source files for the utility libraries used by the sample applications (section 7, page 59).

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 16aiss8ao4.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 16aiss8ao4 in the current directory, and then copies all of the archive's files into this new directory.

```
tar –xzvf 16aiss8ao4.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 57).

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 16aiss8ao4.linux.tar.gz 16aiss8ao4
```

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

```
rm -f /dev/16aiss8ao4.*
```

5. If the automatic startup procedure was adopted (section 1, page 55), then edit the system startup script rc.local and remove the line that invokes the 16AISS8AO4'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 (…/16aiss8ao4/).

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/lib16aiss8ao4_api.so`
Defined and Not Empty	`==== Linking: ../lib/lib16aiss8ao4_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/16aiss8ao4_utils.a`
Defined and Not Empty	`==== Linking: ../lib/16aiss8ao4_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 16AISS8AO4 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 16AISS8AO4 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 16AISS8AO4 specific header files. Therefore, sources may include only this one 16AISS8AO4 header and make files may reference only this one 16AISS8AO4 include directory.

Description	File	Location
Header File	`16aiss8ao4_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 16AISS8AO4 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 16AISS8AO4 static library and only this one 16AISS8AO4 library directory.

Description	File	Location
Static Library	`16aiss8ao4_main.a`	`…/lib/`
	`16aiss8ao4_multi.a`	

NOTE: For applications using the 16AISS8AO4 and no other GSC devices, link the `16aiss8ao4_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 16AISS8AO4 API Library is implemented as a shared library and is thus not linked with the 16AISS8AO4 Main Library. The API Library must be linked with applications by adding the argument `–l16aiss8ao4_api` to the linker command line.

3.2.1. Build

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

1. Change to the directory where the main library resides (`…/lib/`).

2. Remove existing build targets using the below command.

   ```
   make clean
   ```

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

   ```
   make
   ```

3.2.2. System Libraries

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

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

4. API Library

The 16AISS8AO4 API Library is the software interface between user applications and the 16AISS8AO4 device driver. The interface is accessed by including the header file 16aiss8ao4_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	16aiss8ao4_api.h	…/include/
Library File	lib16aiss8ao4_api.so	…/lib/ /usr/lib/ †

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

4.2. Build

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

NOTE: The following steps may require elevated privileges.

1. Change to the directory where the library sources are installed (…/api/).

2. Remove existing build targets using the below command.

   ```
   make clean
   ```

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

   ```
   make
   ```

4.3. Library Use

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

Description	File	Location	Linker Argument
Header File	16aiss8ao4_api.h	…/include/	
Shared Object Library	lib16aiss8ao4_api.so	…/lib/	
		/usr/lib/	-l16aiss8ao4_api

4.4. Macros

The API Library and driver interfaces include the following macros, which are defined in 16aiss8ao4.h.

4.4.1. IOCTL Services

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

4.4.2. Registers

The following gives the complete set of 16AISS8AO4 registers.

4.4.2.1. GSC Registers

The following table gives the complete set of GSC specific 16AISS8AO4 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 *16AISS8AO4 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
`AISS8AO4_GSC_ACR`	Assembly Configuration Register (ACR)
`AISS8AO4_GSC_AVR`	Autocal Values Register (AVR)
`AISS8AO4_GSC_AU0R`	Auxiliary User 0 Register (AU0R)
`AISS8AO4_GSC_AU1R`	Auxiliary User 1 Register (AU1R)
`AISS8AO4_GSC_AU2R`	Auxiliary User 2 Register (AU2R)
`AISS8AO4_GSC_AU3R`	Auxiliary User 3 Register (AU3R)
`AISS8AO4_GSC_BCR`	Board Control Register (BCR)
`AISS8AO4_GSC_BOOR`	Buffered Output Operations Register (BOOR)
`AISS8AO4_GSC_DIOPR`	Digital I/O Port Register (DIOPR)
`AISS8AO4_GSC_IBDR`	Input Buffer Data Register (IBDR)
`AISS8AO4_GSC_IBSR`	Input Buffer Size Register (IBSR)
`AISS8AO4_GSC_IBTR`	Input Buffer Threshold Register (IBTR)
`AISS8AO4_GSC_ICR`	Input Configuration Register (ICR)
`AISS8AO4_GSC_OBDR`	Output Buffer Data Register (OBDR)
`AISS8AO4_GSC_OBSR`	Output Buffer Size Register (OBSR)
`AISS8AO4_GSC_OBTR`	Output Buffer Threshold Register (OBTR)
`AISS8AO4_GSC_OC0DR`	Output Channel 0 Data Register (OC0DR)
`AISS8AO4_GSC_OC1DR`	Output Channel 1 Data Register (OC1DR)
`AISS8AO4_GSC_OC2DR`	Output Channel 2 Data Register (OC2DR)
`AISS8AO4_GSC_OC3DR`	Output Channel 3 Data Register (OC3DR)
`AISS8AO4_GSC_PSR`	Primary Status Register (PSR)
`AISS8AO4_GSC_RAGR`	Rate-A Generator Register (RAGR)
`AISS8AO4_GSC_RBGR`	Rate-B Generator Register (RBGR)
`AISS8AO4_GSC_RCGR`	Rate-C Generator Register (RCGR)

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 `16aiss8ao4_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 `16aiss8ao4_api.h`.

4.5. Data Types

The data types used by the API Library are described with the IOCTL services with which they are used. For additional information refer to section 4.7 (page 25).

4.6. Functions

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

Return Value	Description
`< 0`	This is the value "`(-errno)`" (see `errno.h`).

4.6.1. aiss8ao4_close()

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

Prototype

```
int aiss8ao4_close(int fd);
```

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

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

Example

```
#include <stdio.h>

#include "16aiss8ao4_dsl.h"

int aiss8ao4_close_dsl(int fd)
{
    int errs;
    int ret;

    ret = aiss8ao4_close(fd);

    if (ret)
        printf("ERROR: aiss8ao4_close() returned %d\n", ret);

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

4.6.2. aiss8ao4_init()

This function is the entry point to initializing the 16AISS8AO4 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 aiss8ao4_init(void);
```

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

Example

```
#include <stdio.h>

#include "16aiss8ao4_dsl.h"

int aiss8ao4_init_dsl(void)
{
    int errs;
    int ret;

    ret = aiss8ao4_init();

    if (ret)
        printf("ERROR: aiss8ao4_init() returned %d\n", ret);

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

4.6.3. aiss8ao4_ioctl()

This function is the entry point to performing setup and control operations on a 16AISS8AO4. This function should only be called after a successful open of the respective device. The specific operation performed varies according to the `request` argument. The `request` argument also governs the use and interpretation of the `arg` argument. The set of supported options for the `request` argument consists of the IOCTL services supported by the driver, which are defined in section 4.7 (page 25).

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

Prototype

```
int aiss8ao4_ioctl(int fd, int request, void* arg);
```

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

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

Example

```
#include <stdio.h>

#include "16aiss8ao4_dsl.h"

int aiss8ao4_ioctl_dsl(int fd, int request, void* arg)
{
    int errs;
    int ret;

    ret = aiss8ao4_ioctl(fd, request, arg);

    if (ret)
        printf("ERROR: aiss8ao4_ioctl() returned %d\n", ret);

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

4.6.4. aiss8ao4_open()

This function is the entry point to open a connection to a 16AISS8AO4 board. Before returning, the initialize IOCTL service is called to reset all hardware and software settings to their defaults.

Prototype

```
int aiss8ao4_open(int device, int share, int* fd);
```

<table>
<tr><th>Argument</th><th>Description</th></tr>
<tr><td><code>device</code></td><td>This is the zero-based index of the 16AISS8AO4 to access. †</td></tr>
<tr><td><code>share</code></td><td>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).</td></tr>
<tr><td><code>fd</code></td><td>The device handle is returned here. The pointer cannot be NULL. Values returned are as follows.
<table>
<tr><th>Value</th><th>Description</th></tr>
<tr><td><code>>= 0</code></td><td>This is the handle to use to access the device in subsequent calls.</td></tr>
<tr><td><code>-1</code></td><td>There was an error. The device is not accessible.</td></tr>
</table></td></tr>
</table>

† The index value `-1` can also be given to acquire driver information (section 2.2, page 12).

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

Example

```
#include <stdio.h>

#include "16aiss8ao4_dsl.h"
```

```
int aiss8ao4_open_dsl(int device, int share, int* fd)
{
    int errs;
    int ret;

    ret = aiss8ao4_open(device, share, fd);

    if (ret)
        printf("ERROR: aiss8ao4_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. aiss8ao4_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 12) rather than data from a device.

> **NOTE:** For additional information refer to the Data Transfer Modes section (section 8.4, page 61).

Prototype

```
int aiss8ao4_read(int fd, void* dst, size_t bytes);
```

Argument	Description
`fd`	This is the file descriptor obtained from the open service (section 4.6.4, page 22).
`dst`	The data read is put here.
`bytes`	This is the desired number of bytes to read. When reading from a device, this must be a multiple of four (4).

Return Value	Description
`0` to `bytes`	The operation succeeded. When reading from a device, a value less than `bytes` indicates that the I/O timeout period lapsed (section 4.7.69, page 50) before the entire request could be satisfied.
`< 0`	An error occurred. See error value description above.

Example

```
#include <stdio.h>

#include "16aiss8ao4_dsl.h"

int aiss8ao4_read_dsl(int fd, void* dst, size_t bytes, size_t* qty)
{
    int errs;
    int ret;

    ret = aiss8ao4_read(fd, dst, bytes);

    if (ret < 0)
        printf("ERROR: aiss8ao4_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. aiss8ao4_write()

This function is the entry point to writing data to an open connection. The function writes up to `bytes` bytes.

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

> **NOTE:** For additional information refer to the Data Transfer Modes section (section 8.4, page 61).

Prototype

```
int aiss8ao4_write(int fd, const void* src, size_t bytes);
```

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

Return Value	Description
0 to bytes	The operation succeeded. A value less than bytes indicates that the I/O timeout period lapsed (section 4.7.69, page 50) before the entire request could be satisfied.
< 0	An error occurred. See error value description above.

Example

```
#include <stdio.h>

#include "16aiss8ao4_dsl.h"

int aiss8ao4_write_dsl(int fd, const void* src, size_t bytes, size_t*
qty)
{
    int errs;
    int ret;

    ret = aiss8ao4_write(fd, src, bytes);

    if (ret < 0)
        printf("ERROR: aiss8ao4_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 16AISS8AO4 API Library and device driver implement the following IOCTL services. Each service is described along with the applicable aiss8ao4_ioctl() function arguments.

> **NOTE:** All of the AISS8AO4_IOCTL_AO_XXX IOCTL service requests will produce the EIO error status if the board has no output channels. Refer to the errno.h system header for additional information.

4.7.1. AISS8AO4_IOCTL_AI_BUF_CLEAR

This service clears the content of the input data buffer. It also clears the associated overflow and underflow status bits.

Usage

Argument	Description
request	AISS8AO4_IOCTL_AI_BUF_CLEAR
arg	Unused.

4.7.2. AISS8AO4_IOCTL_AI_BUF_LEVEL

This service reports the number of samples currently in the input data buffer.

Usage

Argument	Description
request	AISS8AO4_IOCTL_AI_BUF_LEVEL
arg	s32*

Valid argument values reported are in the range from zero to 256K (0x40000).

4.7.3. AISS8AO4_IOCTL_AI_BUF_OVERFLOW

This service reports and acts on the occurrence of an input buffer overflow. After clearing the status, this service always reports the updated overflow status.

> **NOTE:** An input buffer overflow occurs when the overall input sample rate exceeds the rate at which the applications is reading data from the input buffer. It is the application's responsibility to ensure that this does not occur.

Usage

Argument	Description
request	AISS8AO4_IOCTL_AI_BUF_OVERFLOW
arg	s32*

Valid argument values provided to the service are as follows.

Value	Description
-1	This requests the current overflow status.
AISS8AO4_BUF_ERROR_CHECK	This requests the current overflow status.
AISS8AO4_BUF_ERROR_CLEAR	This clears an overflow status

Valid argument values reported are as follows.

Value	Description
AISS8AO4_BUF_ERROR_NO	An overflow has not occurred.
AISS8AO4_BUF_ERROR_YES	An overflow has occurred.

4.7.4. AISS8AO4_IOCTL_AI_BUF_THR_LVL

This service adjusts and reports on the input buffer threshold level.

Usage

Argument	Description
request	AISS8AO4_IOCTL_AI_BUF_THR_LVL
arg	s32*

Valid argument values are in the range from zero to 0x7FFFF, or -1 to retrieve the current setting.

4.7.5. AISS8AO4_IOCTL_AI_BUF_THR_STS

This service reports on the input buffer threshold status.

Usage

Argument	Description
request	AISS8AO4_IOCTL_AI_BUF_THR_STS
arg	s32*

Valid argument values reported are as follows.

Value	Description
AISS8AO4_BUF_THR_STS_CLEAR	The number of samples in the input buffer is less than the Threshold Level.
AISS8AO4_BUF_THR_STS_SET	The number of samples in the input buffer equals or exceeds the Threshold Level.

4.7.6. AISS8AO4_IOCTL_AI_BUF_UNDERFLOW

This service reports and acts on the occurrence of an input buffer underflow. After clearing the status, this service always reports the updated underflow status.

> **NOTE:** An input buffer underflow occurs when data is read from the input buffer at a time when the input buffer is empty. The driver and firmware work to ensure that this does not occur.

Usage

Argument	Description
request	AISS8AO4_IOCTL_AI_BUF_UNDERFLOW
arg	s32*

Valid argument values provided to the service are as follows.

Value	Description
-1	This requests the current underflow status.
AISS8AO4_BUF_ERROR_CHECK	This requests the current underflow status.
AISS8AO4_BUF_ERROR_CLEAR	This clears an underflow status

Valid argument values reported are as follows.

Value	Description
AISS8AO4_BUF_ERROR_NO	An underflow has not occurred.
AISS8AO4_BUF_ERROR_YES	An underflow has occurred.

4.7.7. AISS8AO4_IOCTL_AI_BURST_ENABLE

This service controls and reports on the input burst mode selection.

Usage

Argument	Description
request	AISS8AO4_IOCTL_AI_BURST_ENABLE
arg	s32*

Valid argument values are as follows.

Value	Description
-1	This requests the current setting.
AISS8AO4_BURST_ENABLE_NO	Input burst mode is disabled. Input is acquired continuously.
AISS8AO4_BURST_ENABLE_YES	Input burst mode is enabled. Input is acquired in bursts of an application specified size in response to a trigger.

4.7.8. AISS8AO4_IOCTL_AI_BURST_SIZE

This service sets and reports the size of an input burst. The size is specified in scans of all active input channels.

Usage

Argument	Description
request	AISS8AO4_IOCTL_AI_BURST_SIZE
arg	s32*

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

4.7.9. AISS8AO4_IOCTL_AI_BURST_STATUS

This service reports the current input burst status.

Usage

Argument	Description
request	AISS8AO4_IOCTL_AI_BURST_STATUS
arg	s32*

Valid argument values returned are as follows.

Value	Description
AISS8AO4_AI_BURST_STATUS_BUSY	The board is not ready to begin a burst operation.
AISS8AO4_AI_BURST_STATUS_READY	The burst mode is idle and ready to perform a burst operation.

4.7.10. AISS8AO4_IOCTL_AI_CHAN_SEL

This service controls and reports on the set of active input channels. The channels are specified as a mask. If a bit is set, then the channel is enabled. If a bit is clear, then the channel is disabled. Bit 0x01 refers to channel zero and bit 0x80 refers to channel seven.

Usage

Argument	Description
request	AISS8AO4_IOCTL_AI_CHAN_SEL
arg	s32*

Valid argument values are any bitwise combination referencing supported channels. The valid range is from zero to 0xFF for boards with eight input channels and from zero to 0xF for boards with four input channels. The value -1 is also valid and is used to retrieve the current set of active channels.

4.7.11. AISS8AO4_IOCTL_AI_CLOCK_SRC

This service controls and reports on the analog input clock source.

Usage

Argument	Description
request	AISS8AO4_IOCTL_AI_CLOCK_SRC
arg	s32*

Valid argument values are as follows.

Value	Description
-1	This requests the current setting.
AISS8AO4_AI_CLOCK_SRC_EXT	The clock source is the external Analog Input Clock signal from the cable interface. This corresponds to *target* mode operation.
AISS8AO4_AI_CLOCK_SRC_RAG	The clock source is the Rate-A Generator. This corresponds to *initiator* mode operation.

4.7.12. AISS8AO4_IOCTL_AI_ENABLE

This service controls enabling of the input buffer and reports on its current state.

Usage

Argument	Description
request	AISS8AO4_IOCTL_AI_ENABLE
arg	s32*

Valid argument values are as follows.

Value	Description
-1	This requests the current setting.
AISS8AO4_AI_ENABLE_NO	The input buffer is disabled.
AISS8AO4_AI_ENABLE_YES	The input buffer is enabled.

4.7.13. AISS8AO4_IOCTL_AI_MODE

This service adjusts and reports on the analog input mode.

Usage

Argument	Description
request	AISS8AO4_IOCTL_AI_MODE
arg	s32*

Valid argument values are as follows.

Value	Description
-1	This requests the current setting.
AISS8AO4_AI_MODE_DIFF	This refers to differential input operation.
AISS8AO4_AI_MODE_SINGLE	This refers to single ended input operation.
AISS8AO4_AI_MODE_ZERO	This connects the inputs to a zero-reference voltage.
AISS8AO4_AI_MODE_VREF	This connects the inputs to the board's VREF reference source.
AISS8AO4_AI_MODE_AO_0	This connects the inputs to a channel 0 output. *
AISS8AO4_AI_MODE_AO_1	This connects the inputs to a channel 1 output. *
AISS8AO4_AI_MODE_AO_2	This connects the inputs to a channel 2 output. *
AISS8AO4_AI_MODE_AO_3	This connects the inputs to a channel 3 output. *

* These options are valid only for those boards configured with output channels.

4.7.14. AISS8AO4_IOCTL_AI_RANGE_A

Set service controls and reports on the voltage range setting for input channel group A. On boards with eight input channels this includes input channels 0, 1, 4 and 5. On boards with four input channels this includes input channels 0 and 1.

Usage

Argument	Description
request	AISS8AO4_IOCTL_AI_RANGE_A
arg	s32*

Valid argument values are as follows.

Value	Description
-1	This requests the current setting.
AISS8AO4_RANGE_2_5V	This refers to the voltage range of ±2.5 volts.
AISS8AO4_RANGE_5V	This refers to the voltage range of ±5 volts.
AISS8AO4_RANGE_10V	This refers to the voltage range of ±10 volts.

4.7.15. AISS8AO4_IOCTL_AI_RANGE_B

Set service controls and reports on the voltage range setting for input channel group B. On boards with eight input channels this includes input channels 2, 3, 6 and 7. On boards with four input channels this includes input channels 2 and 3.

Usage

Argument	Description
request	AISS8AO4_IOCTL_AI_RANGE_B
arg	s32*

Valid argument values are as follows.

Value	Description
-1	This requests the current setting.
AISS8AO4_RANGE_2_5V	This refers to the voltage range of ±2.5 volts.
AISS8AO4_RANGE_5V	This refers to the voltage range of ±5 volts.
AISS8AO4_RANGE_10V	This refers to the voltage range of ±10 volts.

4.7.16. AISS8AO4_IOCTL_AI_SW_CLOCK

This service initiates a scan of the active input channels.

Usage

Argument	Description
request	AISS8AO4_IOCTL_AI_SW_CLOCK
arg	Unused.

4.7.17. AISS8AO4_IOCTL_AI_SW_TRIGGER

This service initiates a burst mode input acquisition, if burst mode is enabled.

Usage

Argument	Description
request	AISS8AO4_IOCTL_AI_SW_TRIGGER
arg	Unused.

4.7.18. AISS8AO4_IOCTL_AO_ACCESS_MODE

This service selects and reports on the current data output access mode.

Usage

Argument	Description
request	AISS8AO4_IOCTL_AO_ACCESS_MODE
arg	s32*

Valid argument values are as follows.

Value	Description
-1	This requests the current setting.
AISS8AO4_AO_ACCESS_MODE_REG	Analog output is produced by writing to the analog output registers.
AISS8AO4_AO_ACCESS_MODE_FIFO	Analog output is produced via the `write()` service, which buffers data through the analog output FIFO.

4.7.19. AISS8AO4_IOCTL_AO_BUF_CLEAR

This service clears the content of the output data buffer. It also clears the associated data overflow and frame overflow status bits.

Usage

Argument	Description
request	AISS8AO4_IOCTL_AO_BUF_CLEAR
arg	Unused.

4.7.20. AISS8AO4_IOCTL_AO_BUF_DATA_OVER

This service reports and acts on the occurrence of an output buffer data overflow. After clearing the status, this service always reports the updated data overflow status.

> **NOTE:** An output buffer data overflow occurs only when data is written to the output buffer at a time when it is already full. The driver works to ensure that this does not occur.

Usage

Argument	Description
request	AISS8AO4_IOCTL_AO_BUF_DATA_OVER
arg	s32*

Valid argument values provided to the service are as follows.

Value	Description
-1	This requests the current data overflow status.
AISS8AO4_BUF_ERROR_CHECK	This requests the current data overflow status.
AISS8AO4_BUF_ERROR_CLEAR	This clears a data overflow status

Valid argument values reported are as follows.

Value	Description
AISS8AO4_BUF_ERROR_NO	A data overflow has not occurred.
AISS8AO4_BUF_ERROR_YES	A data overflow has occurred.

4.7.21. AISS8AO4_IOCTL_AO_BUF_FRAME_OVER

This service reports and acts on the occurrence of an output buffer frame overflow. After clearing the status, this service always reports the updated frame overflow status.

> **NOTE:** An output buffer frame overflow occurs only when data is written to the output buffer while the buffer is operating in closed/circular mode and at which time access has not been granted. (Refer to AISS8AO4_IOCTL_AO_ACCESS_MODE, section 4.7.18, page 31, and to AISS8AO4_IOCTL_AO_LOAD_REQUEST, page 4.7.34, section 36.) It is the application's responsibility to ensure that this does not occur.

Usage

Argument	Description
request	AISS8AO4_IOCTL_AO_BUF_FRAME_OVER
arg	s32*

Valid argument values provided to the service are as follows.

Value	Description
-1	This requests the current frame overflow status.
AISS8AO4_BUF_ERROR_CHECK	This requests the current frame overflow status.
AISS8AO4_BUF_ERROR_CLEAR	This clears a frame overflow status

Valid argument values reported are as follows.

Value	Description
AISS8AO4_BUF_ERROR_NO	A frame overflow has not occurred.
AISS8AO4_BUF_ERROR_YES	A frame overflow has occurred.

4.7.22. AISS8AO4_IOCTL_AO_BUF_LEVEL

This service reports the number of samples currently in the output data buffer.

Usage

Argument	Description
request	AISS8AO4_IOCTL_AO_BUF_LEVEL
arg	s32*

Valid argument values reported are in the range from zero to 256K (0x40000.

4.7.23. AISS8AO4_IOCTL_AO_BUF_MODE

This service controls and reports on the output buffer's Circular Buffer operating mode.

Usage

Argument	Description
request	AISS8AO4_IOCTL_AO_BUF_MODE
arg	s32*

Valid argument values are as follows.

Value	Description
-1	This requests the current setting.
AISS8AO4_AO_BUF_MODE_CLOSED	The output buffer is closed and operating in the Circular Buffer mode.
AISS8AO4_AO_BUF_MODE_OPEN	The output buffer is open. Output data is not recirculated.

4.7.24. AISS8AO4_IOCTL_AO_BUF_STATUS

This service reports on the relative output buffer fill level.

Usage

Argument	Description
request	AISS8AO4_IOCTL_AO_BUF_STATUS
arg	s32*

Valid argument values returned are as follows.

Value	Description
AISS8AO4_AO_BUF_STATUS_EMPTY	The output buffer is empty.
AISS8AO4_AO_BUF_STATUS_LOW	It isn't empty, but the number of samples in the output buffer is less than the Threshold Level.
AISS8AO4_AO_BUF_STATUS_HIGH	It isn't full, but the number of samples in the output buffer equals or exceeds the Threshold Level.
AISS8AO4_AO_BUF_STATUS_FULL	The output buffer is full.

4.7.25. AISS8AO4_IOCTL_AO_BUF_THR_LVL

This service adjusts and reports on the output buffer threshold level.

Usage

Argument	Description
request	AISS8AO4_IOCTL_AO_BUF_THR_LVL
arg	s32*

Valid argument values are in the range from zero to 0x7FFFF, or `-1` to retrieve the current setting.

4.7.26. AISS8AO4_IOCTL_AO_BUF_THR_STS

This service reports on the output buffer threshold status.

Usage

Argument	Description
request	AISS8AO4_IOCTL_AO_BUF_THR_STS
arg	s32*

Valid argument values reported are as follows.

Value	Description
AISS8AO4_BUF_THR_STS_CLEAR	The number of samples in the output buffer is less than the Threshold Level.
AISS8AO4_BUF_THR_STS_SET	The number of samples in the output buffer equals or exceeds the Threshold Level.

4.7.27. AISS8AO4_IOCTL_AO_BURST_ENABLE

This service controls and reports on the output burst mode selection.

Usage

Argument	Description
request	AISS8AO4_IOCTL_AO_BURST_ENABLE
arg	s32*

Valid argument values are as follows.

Value	Description
-1	This requests the current setting.
AISS8AO4_BURST_ENABLE_NO	Output burst mode is disabled. Output is produced continuously according to the data in the output buffer.
AISS8AO4_BURST_ENABLE_YES	Output burst mode is enabled. Output is produced in bursts according to the data in the output buffer.

4.7.28. AISS8AO4_IOCTL_AO_BURST_STATUS

This service reports the current output burst status.

Usage

Argument	Description
request	AISS8AO4_IOCTL_AO_BURST_STATUS
arg	s32*

Valid argument values are as follows.

Value	Description
AISS8AO4_AO_BURST_STATUS_BUSY	The board is not ready to begin a burst operation.
AISS8AO4_AO_BURST_STATUS_READY	The burst mode is idle and ready to perform a burst operation.

4.7.29. AISS8AO4_IOCTL_AO_CHAN_SEL

This service controls and reports on the set of active output channels. The channels are specified as a mask. If a bit is set, then the channel is enabled. If a bit is clear, then the channel is disabled. Bit `0x01` refers to channel zero and bit `0x8` refers to channel three.

Usage

Argument	Description
request	AISS8AO4_IOCTL_AO_CHAN_SEL
arg	s32*

Valid argument values are any bitwise combination referencing supported channels. The valid range is from zero to `0xF` for boards with four output channels. The value `-1` is also valid and is used to retrieve the current set of active channels.

4.7.30. AISS8AO4_IOCTL_AO_CLOCK_ENABLE

This service controls and reports on the enabled state of the output clock.

Usage

Argument	Description
request	AISS8AO4_IOCTL_AO_CLOCK_ENABLE
arg	s32*

Valid argument values are as follows.

Value	Description
-1	This requests the current state.
AISS8AO4_AO_CLOCK_ENABLE_NO	The output clock is disabled.
AISS8AO4_AO_CLOCK_ENABLE_YES	The output clock is enabled.

4.7.31. AISS8AO4_IOCTL_AO_CLOCK_READY

This service reports on the output buffer’s readiness to clock out data.

Usage

Argument	Description
request	AISS8AO4_IOCTL_AO_CLOCK_READY
arg	s32*

Valid argument values returned are as follows.

Value	Description
AISS8AO4_AO_CLOCK_READY_NO	The board is not ready to clock out data.
AISS8AO4_AO_CLOCK_READY_YES	The board is ready to clock out data.

4.7.32. AISS8AO4_IOCTL_AO_CLOCK_SRC

This service controls and reports on the analog output clock source.

Usage

Argument	Description
request	AISS8AO4_IOCTL_AO_CLOCK_SRC
arg	s32*

Valid argument values are as follows.

Value	Description
-1	Request the current setting.
AISS8AO4_AO_CLOCK_SRC_EXT	The clock source is the external Analog Output Clock signal from the cable interface. This corresponds to *target* mode operation.
AISS8AO4_AO_CLOCK_SRC_RCG	The clock source is the Rate-C Generator. This corresponds to *initiator* mode operation.

4.7.33. AISS8AO4_IOCTL_AO_LOAD_READY

This service reports on the output buffer's readiness to accept output data while the buffer is configured for Circular operation (refer to AISS8AO4_IOCTL_AO_BUF_MODE, section 4.7.23, page 33).

Usage

Argument	Description
request	AISS8AO4_IOCTL_AO_LOAD_READY
arg	s32*

Valid argument values returned are as follows.

Value	Description
AISS8AO4_AO_LOAD_READY_NO	The output buffer is not ready to accept additional output data.
AISS8AO4_AO_LOAD_READY_YES	The output buffer is ready to accept additional output data.

4.7.34. AISS8AO4_IOCTL_AO_LOAD_REQUEST

This service requests access to put data into the output buffer and is applicable only while the buffer is configured for Circular operation (refer to AISS8AO4_IOCTL_AO_BUF_MODE, section 4.7.23, page 33). The service will wait for the request to be granted, but no more than the timeout configured for write() requests (refers to AISS8AO4_IOCTL_TX_IO_TIMEOUT, section 4.7.69, page 50).

Usage

Argument	Description
request	AISS8AO4_IOCTL_AO_LOAD_REQUEST
arg	s32*

Valid argument values returned are as follows.

Value	Description
AISS8AO4_AO_LOAD_REQUEST_NO	The request was not granted within the write timeout period.
AISS8AO4_AO_LOAD_REQUEST_YES	The request was granted.

4.7.35. AISS8AO4_IOCTL_AO_RANGE

Set service controls and reports on the voltage range setting for the analog output channels.

Usage

Argument	Description
request	AISS8AO4_IOCTL_AO_RANGE
arg	s32*

Valid argument values are as follows.

Value	Description
-1	This requests the current setting.
AISS8AO4_RANGE_2_5V	This refers to the voltage range of ±2.5 volts.
AISS8AO4_RANGE_5V	This refers to the voltage range of ±5 volts.
AISS8AO4_RANGE_10V	This refers to the voltage range of ±10 volts.

4.7.36. AISS8AO4_IOCTL_AO_SW_CLOCK

This service initiates a scan of the active output channels.

Usage

Argument	Description
request	AISS8AO4_IOCTL_AO_SW_CLOCK
arg	Unused.

4.7.37. AISS8AO4_IOCTL_AO_SW_TRIGGER

This service initiates an output of burst mode data, if burst mode is enabled.

Usage

Argument	Description
request	AISS8AO4_IOCTL_AO_SW_TRIGGER
arg	Unused.

4.7.38. AISS8AO4_IOCTL_AO_TIMING

Set service controls and reports on the timing at which output data is processed for appearance at the cable interface.

Usage

Argument	Description
request	AISS8AO4_IOCTL_AO_TIMING
arg	s32*

Valid argument values are as follows.

Value	Description
-1	This requests the current setting.
AISS8AO4_AO_TIMING_IMMED	Output appears as soon as the data is provided.
AISS8AO4_AO_TIMING_SIMUL	Output is delayed until the last active channel receives new data.

4.7.39. AISS8AO4_IOCTL_AUTOCAL

This service initiates a device autocalibration cycle. The service returns when the operation completes.

> **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	AISS8AO4_IOCTL_AUTOCAL
arg	Unused.

4.7.40. AISS8AO4_IOCTL_AUTOCAL_AI

This service initiates a device autocalibration cycle of the input side of the board. The service returns when the operation completes.

> **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	AISS8AO4_IOCTL_AUTOCAL_AI
arg	Unused.

4.7.41. AISS8AO4_IOCTL_AUTOCAL_STATUS

Set service reports on the status of the most recent autocalibration cycle.

Usage

Argument	Description
request	AISS8AO4_IOCTL_AUTOCAL_STATUS
arg	s32*

Valid argument values returned are as follows.

Value	Description
AISS8AO4_AUTOCAL_STATUS_ACTIVE	An autocalibration cycle is still in progress.
AISS8AO4_AUTOCAL_STATUS_FAIL	The most recent autocalibration cycle failed.
AISS8AO4_AUTOCAL_STATUS_PASS	The most recent autocalibration cycle passed.

4.7.42. AISS8AO4_IOCTL_BURST_SYNC

This service configures the feature that is used to synchronize the clocking of the first analog input and analog output samples when both are configured for burst operation. This feature requires a specific setup sequence, which is described in the board user manual.

Usage

Argument	Description
request	AISS8AO4_IOCTL_BURST_SYNC
arg	s32*

Valid supported argument values are as follows.

Value	Description
-1	If passed to the API, then the current setting is requested. If returned by the API, then the feature is not supported.
AISS8AO4_BURST_SYNC_DISABLE	This disables the synchronizing option.
AISS8AO4_BURST_SYNC_ENABLE	This enables the synchronizing option.

4.7.43. AISS8AO4_IOCTL_DATA_FORMAT

This service selects and reports on the data encoding format.

Usage

Argument	Description
request	AISS8AO4_IOCTL_DATA_FORMAT
arg	s32*

Valid argument values are as follows.

Value	Description
-1	This requests the current setting.
AISS8AO4_DATA_FORMAT_2S_COMP	This refers to Twos Compliment.
AISS8AO4_DATA_FORMAT_OFF_BIN	This refers to Offset Binary.

4.7.44. AISS8AO4_IOCTL_DIO_DIR_OUT

This service controls and reports on the direction of the cable interface digital I/O signals.

Usage

Argument	Description
request	AISS8AO4_IOCTL_DIO_DIR_OUT
arg	s32*

Valid argument values are as follows.

Value	Description
-1	This requests the current setting.
AISS8AO4_DIO_DIR_OUT_1_IN0_IN	The upper 8-bits are inputs and so are the lower 8-bits.
AISS8AO4_DIO_DIR_OUT_1_IN0_OUT	The upper 8-bits are inputs and the lower 8-bits are outputs.
AISS8AO4_DIO_DIR_OUT_1_OUT_0_IN	The upper 8-bits are outputs and the lower 8-bits are inputs.
AISS8AO4_DIO_DIR_OUT_1_OUT_0_OUT	The upper 8-bits are outputs and so are the lower 8-bits.

4.7.45. AISS8AO4_IOCTL_DIO_READ

This service reads the values at the 16 digital I/O port signals. The bit values returned are the signal values provided to the board for ports configured as inputs or the last value written to the board for ports configured as outputs. The lowest significant bit refers to digital I/O port pin zero.

Usage

Argument	Description
request	AISS8AO4_IOCTL_DIO_READ
arg	s32*

Valid argument values returned are in the range from zero to 0xFFFF.

4.7.46. AISS8AO4_IOCTL_DIO_WRITE

This service writes to and reads from the 16 digital I/O port signals.

Usage

Argument	Description
request	AISS8AO4_IOCTL_DIO_WRITE
arg	s32*

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

4.7.47. AISS8AO4_IOCTL_GEN_A_ENABLE

This service controls and reports on the enable state of the Rate-A Generator.

Usage

Argument	Description
request	AISS8AO4_IOCTL_GEN_A_ENABLE
arg	s32*

Valid argument values are as follows.

Value	Description
-1	This requests the current setting.
AISS8AO4_GEN_ENABLE_NO	The rate generator is disabled.
AISS8AO4_GEN_ENABLE_YES	The rate generator is enabled.

4.7.48. AISS8AO4_IOCTL_GEN_A_NDIV

This service controls and reports on the NDIV divisor for the Rate-A Generator.

Usage

Argument	Description
request	AISS8AO4_IOCTL_GEN_A_NDIV
arg	s32*

Valid argument values are in the range 20 to 0xFFFFFF, and -1 to report on the current setting.

4.7.49. AISS8AO4_IOCTL_GEN_B_ENABLE

This service controls and reports on the enable state of the Rate-B Generator.

Usage

Argument	Description
request	AISS8AO4_IOCTL_GEN_B_ENABLE
arg	s32*

Valid argument values are as follows.

Value	Description
-1	This requests the current setting.
AISS8AO4_GEN_ENABLE_NO	The rate generator is disabled.
AISS8AO4_GEN_ENABLE_YES	The rate generator is enabled.

4.7.50. AISS8AO4_IOCTL_GEN_B_NDIV

This service controls and reports on the NDIV divisor for the Rate-B Generator.

Usage

Argument	Description
request	AISS8AO4_IOCTL_GEN_B_NDIV
arg	s32*

Valid argument values are in the range 20 to 0xFFFFFF, and -1 to report on the current setting.

4.7.51. AISS8AO4_IOCTL_GEN_C_ENABLE

This service controls and reports on the enable state of the Rate-C Generator.

Usage

Argument	Description
request	AISS8AO4_IOCTL_GEN_C_ENABLE
arg	s32*

Valid argument values are as follows.

Value	Description
-1	This requests the current setting.
AISS8AO4_GEN_ENABLE_NO	The rate generator is disabled.
AISS8AO4_GEN_ENABLE_YES	The rate generator is enabled.

4.7.52. AISS8AO4_IOCTL_GEN_C_NDIV

This service controls and reports on the NDIV divisor for the Rate-C Generator.

Usage

Argument	Description
request	AISS8AO4_IOCTL_GEN_C_NDIV
arg	s32*

Valid argument values are in the range 40 to 0xFFFFFF, and -1 to report on the current setting.

4.7.53. AISS8AO4_IOCTL_INITIALIZE

This service initializes the device interface to the state present at the time the device was opened. This includes all software and hardware device settings. The driver waits for completion before returning.

> **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	AISS8AO4_IOCTL_INITIALIZE
arg	Unused.

4.7.54. AISS8AO4_IOCTL_IRQ_ENABLE

This service controls and reports on the enabling of interrupts.

Usage

Argument	Description
request	AISS8AO4_IOCTL_IRQ_ENABLE
arg	s32*

Valid argument values are any combination of the following, or -1 to retrieve the current setting.

Value	Description
AISS8AO4_IRQ_AI_BUF_ERROR	This refers to either an Input Buffer Overflow or an Input Buffer Underflow.
AISS8AO4_IRQ_AI_BURST_DONE	This refers to the completion of an input burst operation.
AISS8AO4_IRQ_AI_BURST_START	This refers to the beginning of an input burst operation.
AISS8AO4_IRQ_AI_CLOCK	This refers to the input clock pulse, regardless of its source.
AISS8AO4_IRQ_AI_THRESH_H2L	This refers to the negation of the input buffer threshold status.
AISS8AO4_IRQ_AI_THRESH_L2H	This refers to the assertion of the input buffer threshold status.
AISS8AO4_IRQ_AO_BUF_ERROR	This refers to an output buffer data overflow or frame overflow.
AISS8AO4_IRQ_AO_BURST_READY	This refers to the assertion of an output burst ready status.
AISS8AO4_IRQ_AO_CLOCK	This refers to the output clock pulse, regardless of its source.
AISS8AO4_IRQ_AO_LD_RDY_H2L	This refers to the negation of the output buffer load ready status.
AISS8AO4_IRQ_AO_LD_RDY_L2H	This refers to the assertion of the output buffer load ready status.
AISS8AO4_IRQ_AO_THRESH_H2L	This refers to the negation of the output buffer threshold status.
AISS8AO4_IRQ_AO_THRESH_L2H	This refers to the assertion of the output buffer threshold status.

AISS8AO4_IRQ_AUTOCAL_DONE	This refers to the completion of an autocalibration cycle.
AISS8AO4_IRQ_DIO_0_L2H	This refers to the assertion of the digital I/O port pin 0 signal.

4.7.55. AISS8AO4_IOCTL_QUERY

This service queries the driver for various pieces of information about the board and the driver. The service works by providing a query option from the table below to the driver. In response, the driver provides the information pertaining to the specified option.

Usage

Argument	Description
request	AISS8AO4_IOCTL_QUERY
arg	s32*

Valid argument values provided to the service are as follows.

Value	Description
AISS8AO4_QUERY_AUTOCAL_AI	This returns the maximum duration of the Analog Input Autocalibration cycle in milliseconds.
AISS8AO4_QUERY_AUTOCAL_MS	This returns the maximum duration of the Autocalibration cycle in milliseconds.
AISS8AO4_QUERY_BURST_SYNC	This indicates if the board supports the feature that synchronizes the clocking of the first analog input and analog output samples when both are configured for burst operation.
AISS8AO4_QUERY_CHANNEL_AI_MAX	This returns the maximum number of input channels supported by the board, which may be more that the board's current configuration.
AISS8AO4_QUERY_CHANNEL_AI_QTY	This returns the actual number of input channels on the current board. If the value returned is -1, then the driver was unable to determine the number of channels.
AISS8AO4_QUERY_CHANNEL_AO_MAX	This returns the maximum number of output channels supported by the board, which may be more that the board's current configuration.
AISS8AO4_QUERY_CHANNEL_AO_QTY	This returns the actual number of output channels on the current board. If the value returned is -1, then the driver was unable to determine the number of channels.
AISS8AO4_QUERY_COUNT	This returns the number of query options supported by the IOCTL service.
AISS8AO4_QUERY_DEVICE_TYPE	This returns the identifier value for the board's type. This should be GSC_DEV_TYPE_16AISS8AO4.
AISS8AO4_QUERY_DMDMA	This indicates if the board supports Demand Mode DMA.
AISS8AO4_QUERY_FGEN_MAX_AI	This returns the maximum supported analog input FGEN value.
AISS8AO4_QUERY_FGEN_MAX_AO	This returns the maximum supported analog output FGEN value.
AISS8AO4_QUERY_FGEN_MIN_AI	This returns the minimum supported analog input FGEN value.
AISS8AO4_QUERY_FGEN_MIN_AO	This returns the minimum supported analog output FGEN value.
AISS8AO4_QUERY_FIFO_SIZE_RX	This returns the size of the input buffer in 32-bit values.
AISS8AO4_QUERY_FIFO_SIZE_TX	This returns the size of the output buffer in 32-bit values.

AISS8AO4_QUERY_FSAMP_MAX_AI	This gives the maximum FSAMP value in S/S for analog input.
AISS8AO4_QUERY_FSAMP_MAX_AO	This gives the maximum FSAMP value in S/S for analog output.
AISS8AO4_QUERY_FSAMP_MIN_AI	This gives the minimum FSAMP value in S/S for analog input.
AISS8AO4_QUERY_FSAMP_MIN_AO	This gives the minimum FSAMP value in S/S for analog output.
AISS8AO4_QUERY_INIT_MS	This returns the duration of a board initialization in milliseconds.
AISS8AO4_QUERY_MASTER_CLOCK	This returns the master clock frequency in hertz.
AISS8AO4_QUERY_NDIV_MAX_AI	This returns the maximum supported NDIV value for analog input.
AISS8AO4_QUERY_NDIV_MAX_AO	This returns the maximum supported NDIV value for analog output.
AISS8AO4_QUERY_NDIV_MIN_AI	This returns the minimum supported NDIV value for analog input.
AISS8AO4_QUERY_NDIV_MIN_AO	This returns the minimum supported NDIV value for analog output.
AISS8AO4_QUERY_RATE_GEN_QTY	This returns the number of Rate Generators on the board.
AISS8AO4_QUERY_REG_AUX_USER	This returns the number of supported Auxiliary User Registers.
AISS8AO4_QUERY_SYS_IO_CFG	This returns the system I/O connector configuration. See below.

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

Value	Description
AISS8AO4_IOCTL_QUERY_ERROR	Either there was a processing error or the query option is unrecognized.

Valid values returned for the `AISS8AO4_QUERY_SYS_IO_CFG` query option are as follows.

Value	Description
AISS8AO4_SYS_IO_CFG_ALT_SMA	This refers to the alternate SMA configuration.
AISS8AO4_SYS_IO_CFG_STANDARD	This refers to the standard configuration.

4.7.56. AISS8AO4_IOCTL_REG_MOD

This service performs a read-modify-write of a 16AISS8AO4 register. This includes only the GSC firmware registers. The PCI and PLX Feature Set Registers are read-only. Refer to `16aiss8ao4.h` for the complete list of GSC firmware registers.

Usage

Argument	Description
request	AISS8AO4_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 bit is modified. If a bit here is zero, then the respective register bit is unmodified.

4.7.57. AISS8AO4_IOCTL_REG_READ

This service reads the value of a 16AISS8AO4 register. This includes the PCI registers, the PLX Feature Set Registers and the GSC firmware registers. Refer to `16aiss8ao4.h` and `gsc_pci9056.h` for the complete list of accessible registers.

Usage

Argument	Description
request	AISS8AO4_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.

4.7.58. AISS8AO4_IOCTL_REG_WRITE

This service writes a value to a 16AISS8AO4 register. This includes only the GSC firmware registers. The PCI and PLX Feature Set Registers are read-only. Refer to `16aiss8ao4.h` for a complete list of the GSC firmware registers.

Usage

Argument	Description
request	AISS8AO4_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.

4.7.59. AISS8AO4_IOCTL_RX_IO_ABORT

This service requests that the current `read()` request be aborted.

Usage

Argument	Description
request	AISS8AO4_IOCTL_RX_IO_ABORT
arg	s32*

Valid argument values returned are as follows.

Value	Description
AISS8AO4_IO_ABORT_NO	The I/O operation was not aborted.
AISS8AO4_IO_ABORT_YES	The I/O operation was aborted.

4.7.60. AISS8AO4_IOCTL_RX_IO_MODE

This service sets the data transfer mode used for data read requests.

Usage

Argument	Description
request	AISS8AO4_IOCTL_RX_IO_MODE
arg	s32*

Valid argument values are as follows.

Value	Description
-1	Retrieve the current setting.
GSC_IO_MODE_BMDMA	Use Block Mode DMA (transfer data immediately).
GSC_IO_MODE_DMDMA	Use Demand Mode DMA (transfer data as it becomes possible to do so).
GSC_IO_MODE_PIO	Use PIO mode, which is repetitive register access. This is the default.

4.7.61. AISS8AO4_IOCTL_RX_IO_OVERFLOW

This service configures the read service to check for an input buffer overflow before performing read operations. Sample data is lost when there is an overflow. If the check is performed and an overflow is detected, then the read service immediately returns an error.

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

NOTE: An input buffer overflow occurs when the overall input sample rate exceeds the rate at which the applications is reading data from the input buffer. It is the application's responsibility to ensure that this does not occur.

Usage

Argument	Description
request	AISS8AO4_IOCTL_RX_IO_OVERFLOW
arg	s32*

Valid argument values are as follows.

Value	Description
-1	Retrieve the current setting.
AISS8AO4_IO_ERROR_CHECK	Perform the check. This is the default.
AISS8AO4_IO_ERROR_IGNORE	Do not perform the check.

4.7.62. AISS8AO4_IOCTL_RX_IO_TIMEOUT

This service selects the time limit for data read requests. The limit is specified in seconds.

Usage

Argument	Description
request	AISS8AO4_IOCTL_RX_IO_TIMEOUT
arg	s32*

Valid argument values are in the range from zero to 3600, -1, and AISS8AO4_IO_TIMEOUT_INFINITE. A value of zero tells the driver not to sleep in order to wait for more data, and should only be used with PIO mode reads. A value of -1 is used to retrieve the current setting. If the option AISS8AO4_IO_TIMEOUT_INFINITE is used, then the driver will wait indefinitely rather than timing out. The default is 10 seconds.

4.7.63. AISS8AO4_IOCTL_RX_IO_UNDERFLOW

This service configures the read service to check for an input buffer underflow before performing read operations. Indeterminate data is obtained when there is an underflow. If the check is performed and an underflow is detected, then the read service immediately returns an error.

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

NOTE: An input buffer underflow occurs when data is read from the input buffer at a time when the input buffer is empty. The driver and firmware work to ensure that this does not occur.

Usage

Argument	Description
request	AISS8AO4_IOCTL_RX_IO_UNDERFLOW
arg	s32*

Valid argument values are as follows.

Value	Description
-1	Retrieve the current setting.
AISS8AO4_IO_ERROR_CHECK	Perform the check. This is the default.
AISS8AO4_IO_ERROR_IGNORE	Do not perform the check.

4.7.64. AISS8AO4_IOCTL_TRIGGER_SRC

This service controls and reports on the burst mode trigger source.

Usage

Argument	Description
request	AISS8AO4_IOCTL_TRIGGER_SRC
arg	s32*

Valid argument values are as follows.

Value	Description
-1	This requests the current setting.
AISS8AO4_TRIGGER_SRC_EXT	The trigger source is the external Trigger I/O signal from the cable interface. This corresponds to *target* mode operation.
AISS8AO4_TRIGGER_SRC_RBG	The trigger source is the Rate-B Generator. This corresponds to *initiator* mode operation.

4.7.65. AISS8AO4_IOCTL_TX_IO_ABORT

This service requests that the current `write()` request be aborted.

Usage

Argument	Description
request	AISS8AO4_IOCTL_TX_IO_ABORT
arg	s32*

Valid argument values returned are as follows.

Value	Description
AISS8AO4_IO_ABORT_NO	The I/O operation was not aborted.
AISS8AO4_IO_ABORT_YES	The I/O operation was aborted.

4.7.66. AISS8AO4_IOCTL_TX_IO_DATA_OVER

This service configures the write service to check for an output buffer data overflow before performing write operations. Sampled data is lost when there is a data overflow. If the check is performed and a data overflow is detected, then the write service immediately returns an error.

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

> **NOTE:** An output buffer data overflow occurs only when data is written to the output buffer at a time when it is already full. The driver works to ensure that this does not occur.

Usage

Argument	Description
request	AISS8AO4_IOCTL_TX_IO_DATA_OVER
arg	s32*

Valid argument values are as follows.

Value	Description
-1	Retrieve the current setting.
AISS8AO4_IO_ERROR_CHECK	Perform the check. This is the default.
AISS8AO4_IO_ERROR_IGNORE	Do not perform the check.

4.7.67. AISS8AO4_IOCTL_TX_IO_FRAME_OVER

This service configures the write service to check for an output buffer frame overflow before performing write operations. Sampled data is lost when there is a frame overflow. If the check is performed and a frame overflow is detected, then the write service immediately returns an error.

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

> **NOTE:** An output buffer frame overflow occurs only when data is written to the output buffer while the buffer is operating in closed/circular mode and at which time access has not been granted. (Refer to AISS8AO4_IOCTL_AO_ACCESS_MODE, section 4.7.18, page 31, and to AISS8AO4_IOCTL_AO_LOAD_REQUEST, page 4.7.34, section 36.) It is the application's responsibility to ensure that this does not occur.

Usage

Argument	Description
request	AISS8AO4_IOCTL_TX_IO_FRAME_OVER
arg	s32*

Valid argument values are as follows.

Value	Description
-1	Retrieve the current setting.
AISS8AO4_IO_ERROR_CHECK	Perform the check. This is the default.
AISS8AO4_IO_ERROR_IGNORE	Do not perform the check.

4.7.68. AISS8AO4_IOCTL_TX_IO_MODE

This service sets the data transfer mode used for data write requests.

Usage

Argument	Description
request	AISS8AO4_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 (transfer data immediately).
GSC_IO_MODE_DMDMA	Use Demand Mode DMA (transfer data as it becomes possible to do so).
GSC_IO_MODE_PIO	Use PIO mode, which is repetitive register access. This is the default.

4.7.69. AISS8AO4_IOCTL_TX_IO_TIMEOUT

This service selects the time limit for data write requests. The limit is specified in seconds.

Usage

Argument	Description
request	AISS8AO4_IOCTL_TX_IO_TIMEOUT
arg	s32*

Valid argument values are in the range from zero to 3600, -1, and AISS8AO4_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 AISS8AO4_IO_TIMEOUT_INFINITE is used, then the driver will wait indefinitely rather than timing out. The default is 10 seconds.

4.7.70. AISS8AO4_IOCTL_WAIT_CANCEL

This service resumes all threads blocked via the AISS8AO4_IOCTL_WAIT_EVENT IOCTL service (section 4.7.71, page 51), 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	AISS8AO4_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 4.7.71.2 on page 52.

gsc	This specifies the set of AISS8AO4_WAIT_GSC_* events whose wait requests are to be cancelled. Refer to section 4.7.71.3 on page 52.
alt	This is unused by the 16AISS8AO4 driver and should be zero.
io	This specifies the set of AISS8AO4_WAIT_IO_* events whose wait requests are to be cancelled. Refer to section 4.7.71.4 on page 53.
timeout_ms	This is unused by wait cancel operations.
count	Upon return this indicates the number of waits that were cancelled.

4.7.71. AISS8AO4_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	AISS8AO4_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 4.7.71.1on page 52.
main	This specifies any number of GSC_WAIT_MAIN_* events that the thread is to wait for. Refer to section 4.7.71.2 on page 52.
gsc	This specifies any number of AISS8AO4_WAIT_GSC_* events that the thread is to wait for. Refer to section 4.7.71.3 on page 52.
alt	This is unused by the 16AISS8AO4 driver and must be zero.
io	This specifies any number of AISS8AO4_WAIT_IO_* events that the thread is to wait for. Refer to section 4.7.71.4 on page 53.
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.

4.7.71.1. 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
GSC_WAIT_FLAG_CANCEL	The wait request was cancelled.
GSC_WAIT_FLAG_DONE	One of the referenced events occurred.
GSC_WAIT_FLAG_TIMEOUT	The timeout period lapsed before a referenced event occurred.

4.7.71.2. 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 16AISS8AO4 and other General Standards products.

Fields	Description
GSC_WAIT_MAIN_DMA0	This refers to the DMA Done interrupt on DMA engine number zero.
GSC_WAIT_MAIN_DMA1	This refers to the DMA Done interrupt on DMA engine number one.
GSC_WAIT_MAIN_GSC	This refers to any of the Interrupt Control/Status Register interrupts.
GSC_WAIT_MAIN_OTHER	This generally refers to an interrupt generated by another device sharing the same interrupt as the 16AISS8AO4.
GSC_WAIT_MAIN_PCI	This refers to any interrupt generated by the 16AISS8AO4.
GSC_WAIT_MAIN_SPURIOUS	This refers to board interrupts which should never be generated.
GSC_WAIT_MAIN_UNKNOWN	This refers to board interrupts whose source could not be identified.

4.7.71.3. 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 Primary Status Register. Applications are responsible for enabling the desired interrupt options. Refer to AISS8AO4_IOCTL_IRQ_ENABLE (section 4.7.54, page 42). The interrupts remain enabled after they occur.

Fields	Description
AISS8AO4_WAIT_GSC_AI_BUF_ERROR	This refers to either an Input Buffer Overflow or an Input Buffer Underflow.
AISS8AO4_WAIT_GSC_AI_BURST_DONE	This refers to the completion of an input burst operation.
AISS8AO4_WAIT_GSC_AI_BURST_START	This refers to the beginning of an input burst operation.
AISS8AO4_WAIT_GSC_AI_CLOCK	This refers to the input clock pulse, regardless of its source.
AISS8AO4_WAIT_GSC_AI_THRESH_H2L	This refers to the negation of the input buffer threshold status.
AISS8AO4_WAIT_GSC_AI_THRESH_L2H	This refers to the assertion of the input buffer threshold status.
AISS8AO4_WAIT_GSC_AO_BUF_ERROR	This refers to an output buffer data overflow or frame overflow.
AISS8AO4_WAIT_GSC_AO_BURST_READY	This refers to the assertion of an output burst ready status.
AISS8AO4_WAIT_GSC_AO_CLOCK	This refers to the output clock pulse, regardless of its source.
AISS8AO4_WAIT_GSC_AO_LD_RDY_H2L	This refers to the negation of the output buffer load ready status.
AISS8AO4_WAIT_GSC_AO_LD_RDY_L2H	This refers to the assertion of the output buffer load ready status.
AISS8AO4_WAIT_GSC_AO_THRESH_H2L	This refers to the negation of the output buffer threshold status.
AISS8AO4_WAIT_GSC_AO_THRESH_L2H	This refers to the assertion of the output buffer threshold status.
AISS8AO4_WAIT_GSC_AUTOCAL_DONE	This refers to the completion of an autocalibration cycle.
AISS8AO4_WAIT_GSC_DIO_0_L2H	This refers to the assertion of the digital I/O port pin 0 signal.

4.7.71.4. `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
`AISS8AO4_WAIT_IO_RX_ABORT`	This refers to read requests which have been aborted.
`AISS8AO4_WAIT_IO_RX_DONE`	This refers to read requests which have been satisfied.
`AISS8AO4_WAIT_IO_RX_ERROR`	This refers to read requests which end due to an error.
`AISS8AO4_WAIT_IO_RX_TIMEOUT`	This refers to read requests which end due to the timeout period lapse.
`AISS8AO4_WAIT_IO_TX_ABORT`	This refers to write requests which have been aborted.
`AISS8AO4_WAIT_IO_TX_DONE`	This refers to write requests which have been satisfied.
`AISS8AO4_WAIT_IO_TX_ERROR`	This refers to write requests which end due to an error.
`AISS8AO4_WAIT_IO_TX_TIMEOUT`	This refers to write requests which end due to the timeout period lapse.

4.7.72. AISS8AO4_IOCTL_WAIT_STATUS

This service counts the number of threads blocked via `AISS8AO4_IOCTL_WAIT_EVENT` IOCTL calls (section 4.7.71, page 51), according to the provided criteria. Any application thread waiting on any of the referenced event options is included in the count.

> **NOTE:** The driver itself makes use of the wait services for various internal operations. Driver initiated waits are not included in the status count.

Usage

Argument	Description
`request`	`AISS8AO4_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 4.7.71.2 on page 52.
`gsc`	This specifies the set of `AISS8AO4_WAIT_GSC_*` events whose wait requests are to be counted. Refer to section 4.7.71.3 on page 52.
`alt`	This is unused by the 16AISS8AO4 driver and should be zero.
`io`	This specifies the set of `AISS8AO4_WAIT_IO_*` events whose wait requests are to be counted. Refer to section 4.7.71.4 on page 53.
`timeout_ms`	This is unused by wait status operations.
`count`	Upon return this indicates the number of threads currently waiting.

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	`16aiss8ao4.h`	
Driver File	`16aiss8ao4.ko` † `16aiss8ao4.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 13), but can be built separately following the below steps.

1. Change to the directory where the driver and its sources are installed (`…/driver/`).

2. Remove existing build targets by issuing the below command.

```
make clean
```

3. Build the driver by issuing the below command.

```
make
```

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

5.3. Startup

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

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

5.3.1. Manual Driver Startup Procedures

Start the driver manually by following the below listed steps.

NOTE: The following steps may require elevated privileges.

1. Change to the directory where the driver sources are installed (…/driver/).

2. Install the driver module and create the device nodes by executing the below command. If any errors are encountered then an appropriate error message will be displayed.

   ```
   ./start
   ```

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

   NOTE: The 16AISS8AO4 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 16aiss8ao4 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/16aiss8ao4.*
   ```

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/16aiss8ao4/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 rwxr-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 rwxr-xr-x. After the directory and file are created as described, reboot to verify boot time loading of the driver. Here is an example of a default version of rc.local.

```
#!/bin/bash

# Add your local content here.
```

5.3.2.2. Default `rc.local` File Permissions

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

5.3.2.3. `systemd` Installations

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

```
systemctl enable rc-local
```

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

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

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

5.3.2.5. SElinux Implications

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

```
yum install setroubleshoot setools
```

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

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

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

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

5.4. Verification

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

1. Verify that the file `/proc/16aiss8ao4` 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/16aiss8ao4
```

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/16aiss8ao4` 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 16aiss8ao4
```

2. Verify that the driver module has been unloaded by issuing the below command. The module name `16aiss8ao4` 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	`16aiss8ao4_dsl.h`	…`/include/`
Library File	`16aiss8ao4_dsl.a`	…`/lib/`

6.2. Build

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

1. Change to the directory where the documentation sources are installed (…`/docsrc/`).

2. Remove existing build targets by issuing the below command.

   ```
   make clean
   ```

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

   ```
   make
   ```

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

6.3. Library Use

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

7. Utilities Source Code

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

For each API function there is a corresponding utility source file with a corresponding utility service. As an example, for the API function `aiss8ao4_open()` there is the utility file `open.c` containing the utility function `aiss8ao4_open_util()`. The naming pattern is as follows: API function `aiss8ao4_xxxx()`, utility file name `xxxx.c`, utility function `aiss8ao4_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 `AISS8AO4_IOCTL_QUERY` there is the utility file `util_query.c` containing the utility function `aiss8ao4_query()`. The naming pattern is as follows: IOCTL code `AISS8AO4_IOCTL_XXXX`, utility file name `util_xxxx.c`, utility function `aiss8ao4_xxxx()`.

7.1. Files

The utility files are summarized in the table below.

Description	Files	Location
Source Files	`*.c`, `*.h` …	…`/utils/`
Header File	`16aiss8ao4_utils.h`	…`/include/`

Library Files	`16aiss8ao4_utils.a`
	`gsc_utils.a`
	`os_utils.a`
	`plx_utils.a`
		…`/lib/`

7.2. Build

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

1. Change to the directory where the utility sources are installed (…`/utils/`).

2. Remove existing build targets by issuing the below command.

   ```
   make clean
   ```

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

   ```
   make
   ```

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

7.3. Library Use

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

8. Operating Information

This section explains some basic operational procedures for using the 16AISS8AO4. This is in no way intended to be a comprehensive guide. This is simply to address a very few issues relating to their use.

8.1. Debugging Aids

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

8.1.1. Device Identification

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

Description	File	Location
Application	`id`	`…/id/`

8.1.2. Detailed Register Dump

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

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

Description	File/Name	Location
Function	`aiss8ao4_reg_list()`	Source File
Source File	`util_reg.c`	`…/utils/`
Header File	`16aiss8ao4_utils.h`	`…/include/`
Library File	`16aiss8ao4_utils.a`	`…/lib/`

8.2. Analog Input Configuration

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

Item	Name/File	Location
Function	`aiss8ao4_config_ai()`	Source File
Source File	`util_config_ai.c`	`…/utils/`
Header File	`16aiss8ao4_utils.h`	`…/include/`
Library File	`16aiss8ao4_utils.a`	`…/lib/`

8.3. 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	aiss8ao4_config_ao()	Source File
Source File	util_config_ao.c	…/utils/
Header File	16aiss8ao4_utils.h	…/include/
Library File	16aiss8ao4_utils.a	…/lib/

8.4. 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.

8.4.1. 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.

8.4.2. BMDMA - Block Mode DMA

This mode is intended for data transfers that do not exceed the size of the corresponding data buffer. In this mode, all data transfer between the PCI interface and the data buffers is done in burst mode. For read requests, the data must already be present in the input buffer before the DMA transfer is initiated. Similarly, for write requests, space must already be available in the output buffer. When these conditions are not met the driver waits until they are before performing the transfer.

8.4.3. DMDMA - Demand Mode DMA

This mode is intended for data transfers that may exceed the size of the 16AISS8AO4 data buffer. In this mode, the DMA transfer is initiated immediately without waiting for data or space to become available. Movement of data occurs over time as the data or space becomes available.

9. Sample Applications

The driver archive includes a variety of sample and test applications located under the `samples` subdirectory. While they are provided without support and without any external documentation, any problems reported will be addressed as time permits. The applications are command line based and produce text output for display on a console. All of the applications are built via the Overall Make Script (section 2.7, page 13), but each may be built individually by changing to its respective directory and issuing the commands "`make clean`" and "`make`". 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 four output channels. The pattern is different for each channel, though they are synchronized at the same modest rate.

9.2. din - Digital Input - …/din/

This application reads the cable's digital I/O signals and reports the values read to the console.

9.3. dout - Digital Output - …/dout/

This application writes a pattern to the cable's digital output lines as it is displayed to the console.

9.4. id - Identify Board - …/id/

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

9.5. 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. rxrate - Receive Rate - …/rxrate/

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

9.7. savedata - Save Acquired Data - …/savedata/

This application configures the board for a modest sample rate, reads a megabyte of data, then saves the data to a hex file.

9.8. signals - Digital Signals - …/signals/

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

9.9. 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
March 14, 2024	Updated to version 4.5.109.50.0.
December 12, 2023	Updated to version 4.5.108.50.0. Updated the kernel support table. Numerous editorial changes. Updated the description of the Input and Output Buffer Clear services. Updated the description of the Autocalibration service. Renamed all Auto_Cal content to Autocal. Renamed all Auto_Cal_Sts content to Autocal_Status.
October 12, 2022	Updated to version 4.4.101.44.0. Expanded automatic startup information. Updated the kernel support table. Added section on environment variables. Added the Burst Sync IOCTL service and query option. Updated the information for the open and close calls.
January 28, 2021	Updated to version 4.3.93.35.0. Updated the kernel support table. Corrected instances of wrong model number. Minor editorial changes. Added a licensing subsection. Added WAIT_EVENT note. Expanded automatic startup information.
May 9, 2019	Updated to version 4.3.85.27.0. Numerous editorial changes.
August 29, 2018	Updated to version 4.2.80.26.0. Updated the inside cover page. Updated the CPU and kernel support section. Minor editorial changes. Updated Block Mode DMA macro and associated information. Document reorganization.
November 29, 2016	Updated to version 4.1.68.18.0. Removed the `built` field from the `/proc/` file. Updated the kernel support table. Renamed the `ain` sample application to `savedata`. Updated the command line arguments for the `aout` sample application. Organized the sample applications alphabetically. Updated the usage of the Wait Event `timeout_ms` field. Updated material on the open call. Added open access mode descriptions. Added support for infinite I/O timeouts. Updated the operating information section. Made various miscellaneous updates. Some document reorganization.
September 14, 2015	Updated to version 4.0.60.8.0. Updated the device node name to include a period before the device index. Removed double underscore that prefaced various data types.
February 27, 2014	Updated to version 3.2.52.0. Updated the kernel support data.
January 9, 2014	Updated to version 3.1.51.0. Updated the kernel support data.
November 14, 2013	Updated to version 3.1.50.0.
July 3, 2013	Updated to version 3.1.45.0. Updated the kernel support data.
July 19, 2012	Updated to version 3.1.39.0. Updated the kernel support data.
February 28, 2012	Updated to version 3.0.36.0. The driver and interface have been overhauled. The utility libraries and sample application set have been updated.
January 9, 2012	Updated to version 2.3.0. Various editorial changes. Updated the CPU and Kernel Support information. Updated the comments for the Initialize IOCTL service. Changed the spelling of various Autocalibration related software items.
November 16, 2009	Updated to version 2.2.2. Updated the kernel support table.
August 25, 2009	Updated to version 2.2.1. Updated the kernel support table.
October 3, 2008	Updated to version 2.2.0. Added an analog input sample application. Removed the extra material covering clearing of the buffers.
September 11, 2008	Updated to version 2.1.0.
August 20, 2008	Updated the driver and the release package. Added more sample applications.
February 13, 2006	Initial release.