16AISS2AO2A2M

16-bit, 2 A/D channels, 2 D/A Channels, 2M S/S/Ch
8-bit Digital I/O, 8-bit Buffered Digital Output, 5M S/S

PMC66-16AISS2AO2A2M

Linux Device Driver
And API Library
User Manual

Manual Revision: July 1, 2019
Driver Release Version 1.1.86.28.0

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

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

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

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

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

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

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

Table of Contents

Table of Figures

1. Introduction

1.1. Purpose

The purpose of this document is to describe the interface to the 16AISS2AO2A2M 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 16AISS2AO2A2M hardware. The API Library and driver interfaces are based on the board’s functionality.

1.2. Acronyms

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

Acronyms	Description
API	Application Programming Interface
BDO	Buffered Digital Output
BMDMA	Block Mode DMA
DMA	Direct Memory Access
DMDMA	Demand Mode DMA
GSC	General Standards Corporation
PIO	Programmed I/O
PMC	PCI Mezzanine Card
RGA	Rate Generator A
RGB	Rate Generator B
RGC	Rate Generator C

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 16AISS2AO2A2M installation directory or any of its subdirectories.
16AISS2AO2A2M	This is used as a general reference to any board supported by this driver.
API Library	This is a library that provides application level access to 16AISS2AO2A2M hardware.
Application	This is the user mode process, which runs in user space with user mode privileges.
Driver	This is the kernel mode 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 16AISS2AO2A2M applications. The overall architecture is illustrated in Figure 1 below.

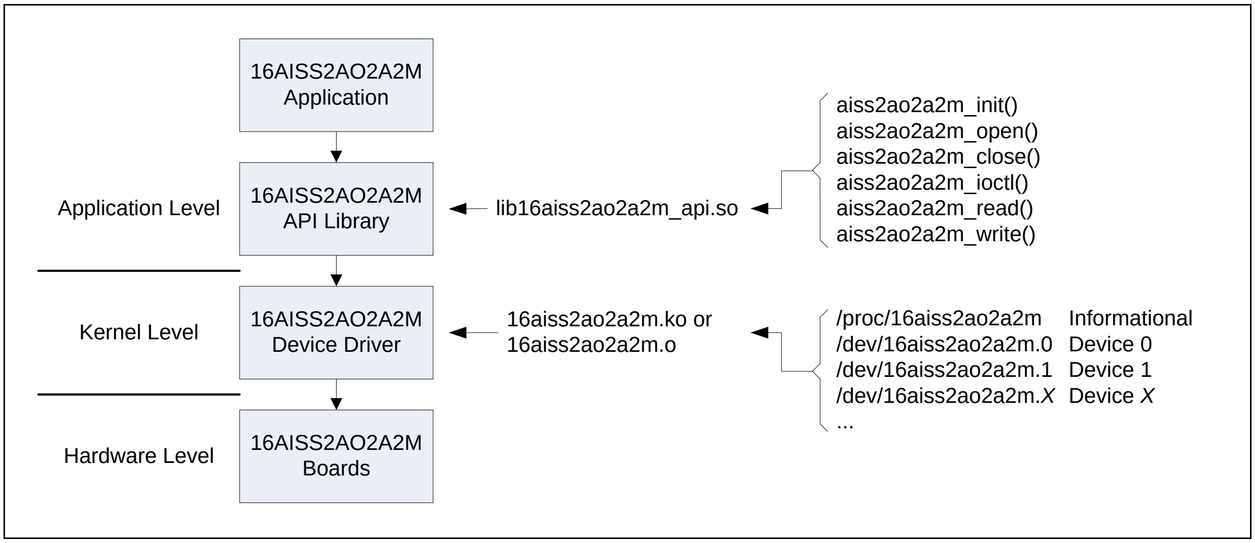
16AISS2AO2A2M Application
Application Level
16AISS2AO2A2M API Library
lib16aiss2ao2a2m_api.so
aiss2ao2a2m_init()
aiss2ao2a2m_open()
aiss2ao2a2m_close()
aiss2ao2a2m_ioctl()
aiss2ao2a2m_read()
aiss2ao2a2m_write()
Kernel Level
16AISS2AO2A2M Device Driver
16aiss2ao2a2m.ko or 16aiss2ao2a2m.o
/proc/16aiss2ao2a2m Informational
/dev/16aiss2ao2a2m.0 Device 0
/dev/16aiss2ao2a2m.1 Device 1
/dev/16aiss2ao2a2m.X Device X
...
Hardware Level
16AISS2AO2A2M Boards

Figure 1 Architectural representation.

1.4.2. API Library

The primary means of accessing 16AISS2AO2A2M boards is via the 16AISS2AO2A2M API Library. This library forms a thin layer between the application and the driver. Additional information is given in section 4 beginning on page 16. With the library, applications are able to open and close a device and, while open, perform I/O control and read operations.

1.4.3. Device Driver

The device driver is the host software that provides a means of communicating directly with 16AISS2AO2A2M 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 16AISS2AO2A2M is a high-performance, 16-bit analog I/O board that incorporates up to two Analog Input channels and up to two Analog Output channels. The host side connection is PCI based and the form factor is according to the model ordered. The board is capable of acquiring and generating data at up to 2M samples per second over each channel. The board has internal clocking that permits the Analog Inputs and Analog Outputs to be independently sampled at rates from 2M samples per second down to nearly two samples per second. Onboard storage permits data buffering of up to 2M Analog Input samples, collectively, for all input channels, between the cable interface and the PCI bus. The Analog Output channels also have 2M sample buffering, collectively, for all output channels. This allows the 16AISS2AO2A2M to sustain continuous throughput over the cable interface independent of the PCI bus interface. The 16AISS2AO2A2M also permits multiple boards to be synchronized so that all boards sample data in unison. In addition, the board includes auto-calibration capability. Furthermore, the board has digital I/O capability, which can be configured as 16 discrete inputs, or as eight discreet outputs with eight outputs buffered through a 256K deep FIFO. The buffered outputs can be clocked from 5M S/S down to nearly two samples per second.

1.6. Reference Material

The following reference material may be of particular benefit in using the 16AISS2AO2A2M. The specifications provide the information necessary for an in depth understanding of the specialized features implemented on this board.

- The applicable *16AISS2AO2A2M 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

2. Installation

2.1. CPU and Kernel Support

The driver is designed to operate with Linux kernel versions 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	x86	
		32-bit	64-bit
5.0.9	Red Hat Fedora Core 30	Yes	Yes
4.18.16	Red Hat Fedora Core 29	Yes	Yes
4.16.3	Red Hat Fedora Core 28	Yes	Yes
4.13.9	Red Hat Fedora Core 27	▨	Yes
4.11.8	Red Hat Fedora Core 26	Yes	Yes
4.8.6	Red Hat Fedora Core 25	Yes	Yes
4.5.5	Red Hat Fedora Core 24	Yes	Yes
4.2.3	Red Hat Fedora Core 23	Yes	Yes
4.0.4	Red Hat Fedora Core 22	Yes	Yes
3.17.4	Red Hat Fedora Core 21	Yes	Yes
3.11.10	Red Hat Fedora Core 20	Yes	Yes
3.9.5	Red Hat Fedora Core 19	Yes	Yes
3.6.10	Red Hat Fedora Core 18	Yes	Yes
3.3.4	Red Hat Fedora Core 17	Yes	Yes
3.1.0	Red Hat Fedora Core 16	Yes	Yes
2.6.38	Red Hat Fedora Core 15	Yes	Yes
2.6.35	Red Hat Fedora Core 14	Yes	Yes
2.6.33	Red Hat Fedora Core 13	Yes	Yes
2.6.31	Red Hat Fedora Core 12	Yes	Yes
2.6.29	Red Hat Fedora Core 11	Yes	Yes
2.6.27	Red Hat Fedora Core 10	Yes	Yes
2.6.25	Red Hat Fedora Core 9	Yes	Yes
2.6.23	Red Hat Fedora Core 8	Yes	Yes
2.6.21	Red Hat Fedora Core 7	Yes	Yes
2.6.18	Red Hat Fedora Core 6	Yes	Yes
2.6.15	Red Hat Fedora Core 5	Yes	Yes
2.6.11	Red Hat Fedora Core 4	Yes	Yes
2.6.9	Red Hat Fedora Core 3	Yes	Yes

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 shipped in source form only.

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

NOTE: The driver has not been tested for SMP operation.

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/16aiss2ao2a2m` file will be "`no`".

2.2. The /proc/ File System

While the driver is running, the text file `/proc/16aiss2ao2a2m` can be read to obtain information about the driver. Each file entry includes an entry name followed immediately by a colon, a space character, and the entry value. Below is an example of what appears in the file, followed by descriptions of each entry.

```
version: 1.1.86.28
32-bit support: yes (native)
boards: 1
models: 16AISS2AO2A2M
```

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.

2.3. File List

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

File	**Description**
`16aiss2ao2a2m.linux.tar.gz`	This archive contains the driver, the API Library and all related files.
`16aiss2ao2a2m_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	**Content**
`16aiss2ao2a2m/`	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 16AISS2AO2A2M API Library sources (section 4, page 16).
…`/docsrc/`	This directory contains the code samples from this document (section 6, page 61).
…`/driver/`	This directory contains the driver and its sources (section 5, page 58).
…`/include/`	This directory contains the include files for the various libraries.
…`/lib/`	This directory contains all of the libraries built from the driver archive.
…`/samples/`	This directory contains the sample applications (section 9, page 65).
…`/utils/`	This directory contains utility sources used by the sample applications (section 7, page 62).

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 `16aiss2ao2a2m.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 `16aiss2ao2a2m` in the current directory, and then copies all of the archive's files into this new directory.

```
tar –xzvf 16aiss2ao2a2m.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.

1. Shutdown the driver as described in section 5.6 on page 60.

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 16aiss2ao2a2m.linux.tar.gz 16aiss2ao2a2m
```

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

```
rm –f /dev/16aiss2ao2a2m.*
```

5. If the automated startup procedure was adopted (section 5.3.2, page 59), then edit the system startup script `rc.local` and remove the line that invokes the 16AISS2AO2A2M'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, and it will also load the driver. 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 (…`/16aiss2ao2a2m/`).

2. Remove existing build targets using the below command line. 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
```

3. Main Interface Files

This section gives general information on the suggested device interface files to use when developing 16AISS2AO2A2M 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 16AISS2AO2A2M 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 16AISS2AO2A2M specific header files. Therefore, sources may include only this one 16AISS2AO2A2M header and make files may reference only this one 16AISS2AO2A2M include directory.

Description	**File**	**Location**
Header File	`16aiss2ao2a2m_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 16AISS2AO2A2M driver archive. For ease of use it is suggested that applications link only the single library file shown below rather than individually linking those libraries identified separately later in this document. Linking this library file pulls in all other pertinent 16AISS2AO2A2M specific static libraries. Therefore, make files may reference only this one 16AISS2AO2A2M static library and only this one 16AISS2AO2A2M library directory.

Description	**File**	**Location**
Static Library	`16aiss2ao2a2m_main.a`	`…/lib/`

> **NOTE**: The 16AISS2AO2A2M API Library is implemented as a shared library and is thus not linked with the 16AISS2AO2A2M Main Library.

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 line.

   ```
   make clean
   ```

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

   ```
   make
   ```

3.2.2. System Libraries

In addition to linking the static library named above, applications may need to also link in additional system libraries as noted below.

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

Real Time	`-lrt`

4. API Library

The 16AISS2AO2A2M API Library is the software interface between user applications and the 16AISS2AO2A2M device driver. The interface is accessed by including the header file `16aiss2ao2a2m_api.h`.

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

4.1. Files

The library source files are summarized in the table below.

File	**Description**
`api/*.c`	These are library source files.
`api/*.h`	These are library header files.
`api/makefile`	This is the library make file.
`api/makefile.dep`	This is an automatically generated make dependency file.
`include/16aiss2ao2a2m_api.h`	This is the library interface header file.
`lib/lib16aiss2ao2a2m_api.so`	This is the API Library shared library file. *

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

4.2. Build

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

> **NOTE**: The API Library shared library is copied to `/usr/lib/`. Therefore, these steps may require elevated privileges.

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

2. Remove existing build targets using the below command line.

   ```
   make clean
   ```

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

   ```
   make
   ```

4.3. Library Use

The library is used at application compile time, at application link time and at application run time. At compile time include the below listed header file in each source file using a component of the library interface. At link time include the below listed linker argument on the linker command line. At link time and at run time the library is found in the directory `/usr/lib/`. (The shared library file is automatically copied to `/usr/lib/` when the library is built.)

File	**Location**	**Linker Argument**
`16aiss2ao2a2m_api.h`	…`/include/`	
`lib16aiss2ao2a2m_api.so`	…`/lib/`	
	`/usr/lib/`	`-l16aiss2ao2a2m_api`

4.4. Macros

The API Library and driver interfaces include the following macros, which are defined in `16aiss2ao2a2m.h`.

4.4.1. IOCTL

The IOCTL macros are documented in section 4.6.5 beginning on page 21.

4.4.2. Registers

The following gives the complete set of 16AISS2AO2A2M registers.

4.4.2.1. GSC Registers

The following tables give the complete set of GSC specific 16AISS2AO2A2M registers. For detailed definitions of these registers refer to the relevant 16AISS2AO2A2M User Manual. Please note that the set of registers supported by any given board may vary by model and firmware version. For the set of supported registers and detailed definitions of these registers please refer to the appropriate *16AISS2AO2A2M User Manual*.

Macro	Description
`AISS2AO2A2M_GSC_ACFGR`	Assembly Configuration Register
`AISS2AO2A2M_GSC_ACR`	Ancillary Control Register
`AISS2AO2A2M_GSC_AIBR`	Analog Input Buffer Register
`AISS2AO2A2M_GSC_AOBR`	Analog Output Buffer Register
`AISS2AO2A2M_GSC_AOC0R`	Analog Output Channel 0 Register
`AISS2AO2A2M_GSC_AOC1R`	Analog Output Channel 1 Register
`AISS2AO2A2M_GSC_AVR`	Autocal Values Register
`AISS2AO2A2M_GSC_BDOBCR`	BDO Buffer Control Register
`AISS2AO2A2M_GSC_BDOBR`	BDO Buffer Register
`AISS2AO2A2M_GSC_BDOBSR`	BDO Buffer Size Register
`AISS2AO2A2M_GSC_BCR`	Board Control Register
`AISS2AO2A2M_GSC_BDORGR`	BDO Rate Generator Register
`AISS2AO2A2M_GSC_BOOR`	Buffered Output Operations Register
`AISS2AO2A2M_GSC_DIOPR`	Digital I/O Port Register
`AISS2AO2A2M_GSC_IBSR`	Input Buffer Size Register
`AISS2AO2A2M_GSC_IBTR`	Input Buffer Threshold Register
`AISS2AO2A2M_GSC_ICR`	Input Configuration Register
`AISS2AO2A2M_GSC_OBSR`	Output Buffer Size Register
`AISS2AO2A2M_GSC_OBTR`	Output Buffer Threshold Register
`AISS2AO2A2M_GSC_PSR`	Primary Status Register
`AISS2AO2A2M_GSC_RAGR`	Rate-A Generator Register
`AISS2AO2A2M_GSC_RBGR`	Rate-B Generator Register
`AISS2AO2A2M_GSC_RCGR`	Rate-C Generator Register

4.4.2.2. PCI Configuration Registers

Access to the PCI registers is seldom required so these registers are not listed here. For the complete list of the PCI register identifiers refer to the driver header file `gsc_pci9056.h`, which is automatically included via `16aiss2ao2a2m.h`.

4.4.2.3. PLX PCI9056 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 `16aiss2ao2a2m.h`.

4.5. Data Types

The data types used by the API Library are described with the IOCTL services with which they are used.

4.6. Functions

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

4.6.1. aiss2ao2a2m_close()

This function is the entry point to close a connection to an open 16AISS2AO2A2M board.

Prototype

```
int aiss2ao2a2m_close(int fd);
```

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

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

Example

```
#include <stdio.h>

#include "16aiss2ao2a2m_dsl.h"

int aiss2ao2a2m_close_dsl(int fd)
{
    int err;
    int ret;

    ret = aiss2ao2a2m_close(fd);

    if (ret)
        printf("ERROR: aiss2ao2a2m_close() returned %d\n", ret);

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

4.6.2. aiss2ao2a2m_init()

This function is the entry point to initializing the 16AISS2AO2A2M 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.

NOTE: This function is not multi-thread safe.

Prototype

```
int aiss2ao2a2m_init(void);
```

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

Example

```
#include <stdio.h>

#include "16aiss2ao2a2m_dsl.h"

int aiss2ao2a2m_init_dsl(void)
{
    int err;
    int ret;

    ret = aiss2ao2a2m_init();

    if (ret)
        printf("ERROR: aiss2ao2a2m_init() returned %d\n", ret);

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

4.6.3. aiss2ao2a2m_ioctl()

This function is the entry point to performing setup and control operations on a 16AISS2AO2A2M board. This function should only be called after a successful open of the respective device. The specific operation performed varies according to the `request` argument. The `request` argument also governs the use and interpretation of the `arg` argument. The set of supported options for the `request` argument consists of the IOCTL services supported by the driver, which are defined in section 4.6.5 beginning on page 21.

Prototype

```
int aiss2ao2a2m_ioctl(int fd, int request, void* arg);
```

Argument	Description
`fd`	This is the file descriptor of the device to access.
`request`	This specifies the desired operation to be performed.
`arg`	This is a `request` specific argument. Refer to the IOCTL services for additional information (section 4.6.5, page 21).

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

Example

```
#include <stdio.h>

#include "16aiss2ao2a2m_dsl.h"

int aiss2ao2a2m_ioctl_dsl(int fd, int request, void *arg)
{
    int err;
    int ret;

    ret = aiss2ao2a2m_ioctl(fd, request, arg);

    if (ret)
        printf("ERROR: aiss2ao2a2m_ioctl() returned %d\n", ret);

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

4.6.4. aiss2ao2a2m_open()

This function is the entry point to open a connection to a 16AISS2AO2A2M board.

Prototype

```
int aiss2ao2a2m_open(int index, int share, int* fd);
```

Argument	Description
`index`	This is the zero based index of the 16AISS2AO2A2M to access. *
`share`	Open the device in Shared Access Mode? If non-zero the device is opened in Shared Access Mode (see below). If zero the device is opened in Exclusive Access Mode (see below).
`fd`	The device handle is returned here. The pointer cannot be NULL. Values returned are as follows. <table><tr><th>Value</th><th>Description</th></tr><tr><td>-1</td><td>There was an error. The device is not accessible.</td></tr><tr><td>>= 0</td><td>This is the handle to use to access the device in subsequent calls.</td></tr></table>

* If the index value is `-1`, then the API Library accesses `/proc/16aiss2ao2a2m`.

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

Example

```
#include <stdio.h>

#include "16aiss2ao2a2m_dsl.h"
```

```
int aiss2ao2a2m_open_dsl(int index, int share, int* fd)
{
    int err;
    int ret;

    ret = aiss2ao2a2m_open(index, share, fd);

    if (ret)
        printf("ERROR: aiss2ao2a2m_open() returned %d\n", ret);

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

4.6.4.1. Access Modes

Shared Access Mode:

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

Exclusive Access Mode:

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

4.6.5. aiss2ao2a2m_read_ai()

This function is the entry point to reading Analog Input data from an open 16AISS2AO2A2M. This function should only be called after a successful open of the respective device. The function reads up to `bytes` bytes from the board. The return value is the number of bytes actually read.

Prototype

```
int aiss2ao2a2m_read_ai(int fd, void *dst, size_t bytes);
```

Argument	Description
`fd`	This is the file descriptor of the device to access.
`dst`	The data read will be put here.
`bytes`	This is the desired number of bytes to read. 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 request timed out.
`< 0`	An error occurred. This is the negative of `errno` from `errno.h`.

Example

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

```
#include "16aiss2ao2a2m_dsl.h"

int aiss2ao2a2m_read_ai_dsl(int fd, void* dst, size_t bytes)
{
    int ret;

    ret = aiss2ao2a2m_read_ai(fd, dst, bytes);

    if (ret < 0)
        printf("ERROR: aiss2ao2a2m_read_ai() returned %d\n", ret);

    return(ret);
}
```

4.6.6. aiss2ao2a2m_write_ao()

This function is the entry point to writing Analog Output data to an open 16AISS2AO2A2M. This function should only be called after a successful open of the respective device. The function writes up to `bytes` bytes to the board. The return value is the number of bytes actually written.

Prototype

```
int aiss2ao2a2m_write_ao(int fd, const void *src, size_t bytes);
```

Argument	**Description**
`fd`	This is the file descriptor of the device to access.
`src`	The data written come 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 request timed out.
`< 0`	An error occurred. This is the negative of `errno` from `errno.h`.

Example

```
#include <stdio.h>

#include "16aiss2ao2a2m_dsl.h"

int aiss2ao2a2m_write_ao_dsl(int fd, const void* src, size_t bytes)
{
    int ret;

    ret = aiss2ao2a2m_write_ao(fd, src, bytes);

    if (ret < 0)
        printf("ERROR: aiss2ao2a2m_write_ao() returned %d\n", ret);

    return(ret);
}
```

4.6.7. aiss2ao2a2m_write_bdo()

This function is the entry point to writing Buffered Digital Output data to an open 16AISS2AO2A2M. This function should only be called after a successful open of the respective device. The function writes up to `bytes` bytes to the board. The return value is the number of bytes actually written.

> **NOTE**: Output data values consist of 32-bits, with the lower eight bits consisting of the digital data of interest. The upper 24 bits should be zero.

Prototype

```
int aiss2ao2a2m_write_bdo(int fd, const void *src, size_t bytes);
```

Argument	**Description**
`fd`	This is the file descriptor of the device to access.
`src`	The data written come 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 request timed out.
`< 0`	An error occurred. This is the negative of `errno` from `errno.h`.

Example

```
#include <stdio.h>

#include "16aiss2ao2a2m_dsl.h"

int aiss2ao2a2m_write_bdo_dsl(int fd, const void* src, size_t bytes)
{
    int ret;

    ret = aiss2ao2a2m_write_bdo(fd, src, bytes);

    if (ret < 0)
        printf("ERROR: aiss2ao2a2m_write_bdo() returned %d\n", ret);

    return(ret);
}
```

4.7. IOCTL Services

The 16AISS2AO2A2M API Library supports the following IOCTL services. Each service is described along with the applicable IOCTL function arguments. Unless otherwise stated the return value definitions are those defined for the `aiss2ao2a2m_ioctl()` function call and any error codes are accessed via `errno`.

4.7.1. AISS2AO2A2M_IOCTL_AI_BUF_CLEAR

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

Usage

Argument	Description
`request`	`AISS2AO2A2M_IOCTL_AI_BUF_CLEAR`
`arg`	Not used.

4.7.2. AISS2AO2A2M_IOCTL_AI_BUF_ENABLE

This service enables or disables Analog Input buffer data reception.

Usage

Argument	Description
`request`	`AISS2AO2A2M_IOCTL_AI_BUF_ENABLE`
`arg`	`s32*`

Valid argument values are as follows.

Value	Description
`-1`	This option retrieves the current setting.
`AISS2AO2A2M_AI_BUF_ENABLE_NO`	This option disables the Analog Input buffer.
`AISS2AO2A2M_AI_BUF_ENABLE_YES`	This option enables the Analog Input buffer.

4.7.3. AISS2AO2A2M_IOCTL_AI_BUF_LEVEL

This service returns the current number of 32-bit values in the Analog Input buffer.

Usage

Argument	Description
`request`	`AISS2AO2A2M_IOCTL_AI_BUF_LEVEL`
`arg`	`s32*`

The value returned will be from zero to 2M (2,097,152).

4.7.4. AISS2AO2A2M_IOCTL_AI_BUF_OVERFLOW

This service operates on the Analog Input Buffer Overflow status.

Usage

Argument	Description
`request`	`AISS2AO2A2M_IOCTL_AI_BUF_OVERFLOW`
`arg`	`s32*`

Valid argument values supplied to the service are as follows.

Value	Description
`-1`	Retrieve the current state.
`AISS2AO2A2M_AI_BUF_OVERFLOW_CHECK`	Check the overflow status.
`AISS2AO2A2M_AI_BUF_OVERFLOW_CLEAR`	Clear the overflow status.

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

Value	Description
`AISS2AO2A2M_AI_BUF_OVERFLOW_NO`	The buffer has not experienced an overflow condition.
`AISS2AO2A2M_AI_BUF_OVERFLOW_YES`	The buffer has experienced an overflow condition.

4.7.5. AISS2AO2A2M_IOCTL_AI_BUF_UNDERFLOW

This service operates on the Analog Input Buffer Underflow status.

Usage

Argument	Description
`request`	`AISS2AO2A2M_IOCTL_AI_BUF_UNDERFLOW`
`arg`	`s32*`

Valid argument values supplied to the service are as follows.

Value	Description
`-1`	Retrieve the current state.
`AISS2AO2A2M_AI_BUF_UNDERFLOW_CHECK`	Check the underflow status.
`AISS2AO2A2M_AI_BUF_UNDERFLOW_CLEAR`	Clear the underflow status.

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

Value	Description
`AISS2AO2A2M_AI_BUF_UNDERFLOW_NO`	The buffer has not experienced an underflow condition.
`AISS2AO2A2M_AI_BUF_UNDERFLOW_YES`	The buffer has experienced an underflow condition.

4.7.6. AISS2AO2A2M_IOCTL_AI_BURST_ENABLE

This service enables and disables Analog Input bursting.

Usage

Argument	Description
`request`	`AISS2AO2A2M_IOCTL_AI_BURST_ENABLE`
`arg`	`s32*`

Valid argument values are as follows.

Value	Description
`-1`	Retrieve the current setting.
`AISS2AO2A2M_AI_BURST_ENABLE_NO`	This option disables Analog Input bursting.
`AISS2AO2A2M_AI_BURST_ENABLE_YES`	This option enables Analog Input bursting.

4.7.7. AISS2AO2A2M_IOCTL_AI_BURST_SIZE

This service configures the size of a single Analog Input burst (the count is in scans, which is an A/D conversion of all active input channels).

Usage

Argument	Description
`request`	`AISS2AO2A2M_IOCTL_AI_BURST_SIZE`
`arg`	`s32*`

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

4.7.8. AISS2AO2A2M_IOCTL_AI_BURST_STATUS

This service reports on the board's Analog Input burst status.

Usage

Argument	Description
`request`	`AISS2AO2A2M_IOCTL_AI_BURST_STATUS`
`arg`	`s32*`

The value returned will be one of the following.

Value	Description
`AISS2AO2A2M_AI_BURST_STATUS_BUSY`	The board is not ready to start an Analog Input burst operation.
`AISS2AO2A2M_AI_BURST_STATUS_IDLE`	The board is ready to start an Analog Input burst operation.

4.7.9. AISS2AO2A2M_IOCTL_AI_CHAN_0_RANGE

This service configures the voltage range for Analog Input channel zero.

Usage

Argument	Description
`request`	`AISS2AO2A2M_IOCTL_AI_CHAN_0_RANGE`
`arg`	`s32*`

Valid argument values are as follows.

Value	Description
`-1`	Retrieve the current setting.
`AISS2AO2A2M_AI_CHAN_0_RANGE_2_5V`	Set the channel voltage range to ±2.5 volts.
`AISS2AO2A2M_AI_CHAN_0_RANGE_5V`	Set the channel voltage range to ±5 volt.
`AISS2AO2A2M_AI_CHAN_0_RANGE_10V`	Set the channel voltage range to ±10 volts.

4.7.10. AISS2AO2A2M_IOCTL_AI_CHAN_1_RANGE

This service configures the voltage range for Analog Input channel one.

Usage

Argument	Description
`request`	`AISS2AO2A2M_IOCTL_AI_CHAN_1_RANGE`
`arg`	`s32*`

Valid argument values are as follows.

Value	Description
`-1`	Retrieve the current setting.
`AISS2AO2A2M_AI_CHAN_1_RANGE_2_5V`	Set the channel voltage range to ±2.5 volts.
`AISS2AO2A2M_AI_CHAN_1_RANGE_5V`	Set the channel voltage range to ±5 volt.

`AISS2AO2A2M_AI_CHAN_1_RANGE_10V`	Set the channel voltage range to ±10 volts.

4.7.11. AISS2AO2A2M_IOCTL_AI_CHAN_SEL

This service configures the set of active Analog Input channels. If a bit is set, then that channel is enabled. If a bit is clear, then that channel is disabled.

Usage

Argument	Description
`request`	`AISS2AO2A2M_IOCTL_AI_CHAN_SEL`
`arg`	`s32*`

Valid argument values are from zero to `0x3` for two channel boards, from zero to `0x1` for single channel boards, or `-1` to retrieve the current setting.

4.7.12. AISS2AO2A2M_IOCTL_AI_IO_ABORT

This service aborts an ongoing `aiss2ao2a2m_read_ai()` request.

Usage

Argument	Description
`request`	`AISS2AO2A2M_IOCTL_AI_IO_ABORT`
`arg`	`s32*`

The results are reported as one of the following values.

Value	Description
`AISS2AO2A2M_AI_IO_ABORT_NO`	A read request was not aborted as none were ongoing.
`AISS2AO2A2M_AI_IO_ABORT_YES`	A read request was aborted.

4.7.13. AISS2AO2A2M_IOCTL_AI_IO_MODE

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

Usage

Argument	Description
`request`	`AISS2AO2A2M_IOCTL_AI_IO_MODE`
`arg`	`s32*`

Valid argument values are as follows. See also section 8.4 on page 63.

Value	Description
`-1`	Retrieve the current setting.
`GSC_IO_MODE_BMDMA`	Use Block Mode DMA.
`GSC_IO_MODE_DMDMA`	Use Demand Mode DMA.
`GSC_IO_MODE_PIO`	Use PIO mode. This is the default.

4.7.14. AISS2AO2A2M_IOCTL_AI_IO_OVERFLOW

This service configures the Analog Input read service to check for an Analog Input buffer overflow before performing read operations. Sampled 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.

Usage

Argument	Description
`request`	`AISS2AO2A2M_IOCTL_AI_IO_OVERFLOW`
`arg`	`s32*`

Valid argument values are as follows.

Value	Description
`-1`	Retrieve the current setting.
`AISS2AO2A2M_AI_IO_OVERFLOW_CHECK`	Perform the check. This is the default.
`AISS2AO2A2M_AI_IO_OVERFLOW_IGNORE`	Do not perform the check.

4.7.15. AISS2AO2A2M_IOCTL_AI_IO_TIMEOUT

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

Usage

Argument	Description
`request`	`AISS2AO2A2M_IOCTL_AI_IO_TIMEOUT`
`arg`	`s32*`

Valid argument values are as follows.

Value	Description
`-1`	Retrieve the current setting.
`0`	Do not sleep to wait for more data.
`1` to `3600`	Timeout after this number of seconds.
`AISS2AO2A2M_AI_IO_TIMEOUT_INFINITE`	Wait indefinitely.

4.7.16. AISS2AO2A2M_IOCTL_AI_IO_UNDERFLOW

This service configures the Analog Input read service to check for an Analog Input buffer underflow before performing read operations. Indeterminate data is returned 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.

Usage

Argument	Description
`request`	`AISS2AO2A2M_IOCTL_AI_IO_UNDERFLOW`
`arg`	`s32*`

Valid argument values are as follows.

Value	Description
`-1`	Retrieve the current setting.
`AISS2AO2A2M_AI_IO_UNDERFLOW_CHECK`	Perform the check. This is the default.
`AISS2AO2A2M_AI_IO_UNDERFLOW_IGNORE`	Do not perform the check.

4.7.17. AISS2AO2A2M_IOCTL_AI_MODE

This service configures the board’s Analog Input Mode.

Usage

Argument	Description
`request`	`AISS2AO2A2M_IOCTL_AI_MODE`
`arg`	`s32*`

Valid argument values are as follows.

Value	Description
`-1`	Retrieve the current setting.
`AISS2AO2A2M_AI_MODE_AO_1`	Route Analog Output channel zero to the analog inputs.
`AISS2AO2A2M_AI_MODE_AO_0`	Route Analog Output channel one to the analog inputs.
`AISS2AO2A2M_AI_MODE_DIFF`	Configure the analog input channels for differential operation.
`AISS2AO2A2M_AI_MODE_SINGLE`	Configure the analog input channels for single-ended operation.
`AISS2AO2A2M_AI_MODE_VREF`	Configure the analog input channels for +VREF input testing
`AISS2AO2A2M_AI_MODE_ZERO`	Configure the analog input channels for Zero input testing

4.7.18. AISS2AO2A2M_IOCTL_AI_SW_CLOCK

This service initiates a manual clock cycle for Analog Input sampling. The driver returns immediately and does not wait for completion.

Usage

Argument	Description
`request`	`AISS2AO2A2M_IOCTL_AI_SW_CLOCK`
`arg`	Not used.

4.7.19. AISS2AO2A2M_IOCTL_AI_SW_TRIGGER

This service initiates a manual trigger cycle for Analog Input bursting. The driver returns immediately and does not wait for completion.

Usage

Argument	Description
`request`	`AISS2AO2A2M_IOCTL_AI_SW_TRIGGER`

`arg`	Not used.

4.7.20. AISS2AO2A2M_IOCTL_AI_THRESH_LVL

This service configures the Analog Input buffer threshold level.

Usage

Argument	Description
`request`	`AISS2AO2A2M_IOCTL_AI_THRESH_LVL`
`arg`	`s32*`

Valid argument values are from zero to `0x3FFFFF`, and `-1`. A value of `-1` will return the current threshold level setting.

4.7.21. AISS2AO2A2M_IOCTL_AI_THRESH_STS

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

Usage

Argument	Description
`request`	`AISS2AO2A2M_IOCTL_AI_THRESH_STS`
`arg`	`s32*`

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

Value	Description
`AISS2AO2A2M_AI_THRESH_STS_CLEAR`	The buffer contains Threshold Level number of data items, or fewer.
`AISS2AO2A2M_AI_THRESH_STS_SET`	The buffer contains more than Threshold Level number of data items.

4.7.22. AISS2AO2A2M_IOCTL_AO_ACCESS_MODE

This service configures the Analog Output access mode.

Usage

Argument	Description
`request`	`AISS2AO2A2M_IOCTL_AO_ACCESS_MODE`
`arg`	`s32*`

Valid argument values are as follows.

Value	Description
`-1`	Retrieve the current setting.
`AISS2AO2A2M_AO_ACCESS_MODE_FIFO`	Analog Output is generated via the Analog Output FIFO, which is access via the `aiss2ao2a2m_write_ao()` API call.
`AISS2AO2A2M_AO_ACCESS_MODE_REG`	Analog Output is generated via individual register writes to the Analog Output channel registers.

4.7.23. AISS2AO2A2M_IOCTL_AO_BUF_CLEAR

This service immediately clears the current content from the Analog Output buffer. This service does not halt output sampling.

Usage

Argument	**Description**
`request`	`AISS2AO2A2M_IOCTL_AO_BUF_CLEAR`
`arg`	Not used.

4.7.24. AISS2AO2A2M_IOCTL_AO_BUF_LEVEL

This service returns the current number of 32-bit data values in the Analog Output buffer.

Usage

Argument	**Description**
`request`	`AISS2AO2A2M_IOCTL_AO_BUF_LEVEL`
`arg`	`s32*`

The value returned will be from zero to 2M (2,097,152).

4.7.25. AISS2AO2A2M_IOCTL_AO_BUF_LOAD_REQ

This service initiates a request to load data into a closed Analog Output buffer. The driver returns immediately without waiting for the request to be granted.

Usage

Argument	**Description**
`request`	`AISS2AO2A2M_IOCTL_AO_BUF_LOAD_REQ`
`arg`	Not used.

4.7.26. AISS2AO2A2M_IOCTL_AO_BUF_LOAD_STS

This service returns the load status of the Analog Output buffer in response to the load request (see previous IOCTL service, `AISS2AO2A2M_IOCTL_AO_BUF_LOAD_REQ`).

Usage

Argument	**Description**
`request`	`AISS2AO2A2M_IOCTL_AO_BUF_LOAD_STS`
`arg`	`s32*`

Valid argument values are as follows.

Value	**Description**
`AISS2AO2A2M_AO_BUF_LOAD_STS_BUSY`	The Analog Output buffer is not ready for additional data to be loaded.
`AISS2AO2A2M_AO_BUF_LOAD_STS_READY`	The Analog Output buffer is ready for additional data to be loaded.

4.7.27. AISS2AO2A2M_IOCTL_AO_BUF_MODE

This service configures the Analog Output buffer data output mode.

Usage

Argument	Description
`request`	`AISS2AO2A2M_IOCTL_AO_BUF_MODE`
`arg`	`s32*`

Valid argument values are as follows.

Value	Description
`-1`	Retrieve the current setting.
`AISS2AO2A2M_AO_BUF_MODE_CIRC`	Data is fed back to the input after being read from the output.
`AISS2AO2A2M_AO_BUF_MODE_OPEN`	Data does not get recycled.

4.7.28. AISS2AO2A2M_IOCTL_AO_BUF_OVER_DATA

This service operates on Analog Output buffer data overflows, which are overflows that occur when the buffer is configured for *open* operation (see `AISS2AO2A2M_IOCTL_AO_BUF_MODE` above).

Usage

Argument	Description
`request`	`AISS2AO2A2M_IOCTL_AO_BUF_OVER_DATA`
`arg`	`s32*`

Valid argument values supplied to the service are as follows.

Value	Description
`-1`	Retrieve the current state.
`AISS2AO2A2M_AO_BUF_OVER_DATA_CHECK`	Check the overflow status.
`AISS2AO2A2M_AO_BUF_OVER_DATA_CLEAR`	Clear the overflow status.

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

Value	Description
`AISS2AO2A2M_AO_BUF_OVER_DATA_NO`	The buffer has not experienced an overflow condition.
`AISS2AO2A2M_AO_BUF_OVER_DATA_YES`	The buffer has experienced an overflow condition.

4.7.29. AISS2AO2A2M_IOCTL_AO_BUF_OVER_FRAME

This service operates on Analog Output buffer frame overflows, which are overflows that occur when the buffer is configured for *circ*, or *circular*, operation (see `AISS2AO2A2M_IOCTL_AO_BUF_MODE` above).

Usage

Argument	Description
`request`	`AISS2AO2A2M_IOCTL_AO_BUF_OVER_FRAME`
`arg`	`s32*`

Valid argument values supplied to the service are as follows.

Value	**Description**
`-1`	Retrieve the current state.
`AISS2AO2A2M_AO_BUF_OVER_FRAME_CHECK`	Check the overflow status.
`AISS2AO2A2M_AO_BUF_OVER_FRAME_CLEAR`	Clear the overflow status.

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

Value	**Description**
`AISS2AO2A2M_AO_BUF_OVER_FRAME_NO`	The buffer has not experienced an overflow condition.
`AISS2AO2A2M_AO_BUF_OVER_FRAME_YES`	The buffer has experienced an overflow condition.

4.7.30. AISS2AO2A2M_IOCTL_AO_BUF_STATUS

This service reports on the fill status of the Analog Output buffer.

Usage

Argument	**Description**
`request`	`AISS2AO2A2M_IOCTL_AO_BUF_STATUS`
`arg`	`s32*`

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

Value	**Description**
`AISS2AO2A2M_AO_BUF_STATUS_EMPTY`	The buffer is empty.
`AISS2AO2A2M_AO_BUF_STATUS_AT_LOW`	The buffer fill level is equal to at below the buffer threshold level, though not empty.
`AISS2AO2A2M_AO_BUF_STATUS_ABOVE`	The buffer fill level is above the buffer threshold level, though not full.
`AISS2AO2A2M_AO_BUF_STATUS_FULL`	The buffer is full.

4.7.31. AISS2AO2A2M_IOCTL_AO_BURST_ENABLE

This service enables and disables Analog Output bursting.

Usage

Argument	**Description**
`request`	`AISS2AO2A2M_IOCTL_AO_BURST_ENABLE`
`arg`	`s32*`

Valid argument values are as follows.

Value	**Description**
`-1`	Retrieve the current setting.
`AISS2AO2A2M_AO_BURST_ENABLE_NO`	This option disables Analog Output bursting.
`AISS2AO2A2M_AO_BURST_ENABLE_YES`	This option enables Analog Output bursting.

4.7.32. AISS2AO2A2M_IOCTL_AO_BURST_STATUS

This service reports on the board’s Analog Output burst status.

Usage

Argument	Description
`request`	`AISS2AO2A2M_IOCTL_AO_BURST_STATUS`
`arg`	`s32*`

The value returned will be one of the following.

Value	Description
`AISS2AO2A2M_AO_BURST_STATUS_BUSY`	The board is not ready to start an Analog Output burst operation.
`AISS2AO2A2M_AO_BURST_STATUS_READY`	The board is ready to start an Analog Output burst operation.

4.7.33. AISS2AO2A2M_IOCTL_AO_BURST_SYNC_AI

This service configures the synchronization between Analog Output burst operations with Analog Input burst operations.

Usage

Argument	Description
`request`	`AISS2AO2A2M_IOCTL_AO_BURST_SYNC_AI`
`arg`	`s32*`

The value returned will be one of the following.

Value	Description
`AISS2AO2A2M_AO_BURST_SYNC_AI_NO`	Analog Output and Analog Input bursts are not forced to become synchronized.
`AISS2AO2A2M_AO_BURST_SYNC_AI_YES`	Analog Output and Analog Input bursts are forced to become synchronized.

4.7.34. AISS2AO2A2M_IOCTL_AO_CHAN_SEL

This service configures the set of active Analog Output channels. If a bit is set, then that channel is enabled. If a bit is clear, then that channel is disabled.

Usage

Argument	Description
`request`	`AISS2AO2A2M_IOCTL_AO_CHAN_SEL`
`arg`	`s32*`

Valid argument values are from zero to `0x3` for two output channel boards and `0x0` for boards with no output channels. To request the current setting pass in the value `-1`.

4.7.35. AISS2AO2A2M_IOCTL_AO_CLOCK_ENABLE

This service enables or disables the Analog Output clock.

Usage

Argument	Description
`request`	`AISS2AO2A2M_IOCTL_AO_CLOCK_ENABLE`
`arg`	`s32*`

Valid argument values are as follows.

Value	Description
`-1`	This option retrieves the current setting.
`AISS2AO2A2M_AO_CLOCK_ENABLE_NO`	This option disables the Analog Output clock.
`AISS2AO2A2M_AO_CLOCK_ENABLE_YES`	This option enables the Analog Output clock.

4.7.36. AISS2AO2A2M_IOCTL_AO_CLOCK_SOURCE

This service configures the Analog Output clock source selection.

Usage

Argument	Description
`request`	`AISS2AO2A2M_IOCTL_AO_CLOCK_SOURCE`
`arg`	`s32*`

Valid argument values are as follows.

Value	Description
`-1`	Retrieve the current setting.
`AISS2AO2A2M_AO_CLOCK_SOURCE_EXT`	This selects external clocking.
`AISS2AO2A2M_AO_CLOCK_SOURCE_INT`	This selects internal clocking from the Rate-C Generator.

4.7.37. AISS2AO2A2M_IOCTL_AO_CLOCK_STATUS

This service reports on the board's Analog Output clock readiness status.

Usage

Argument	Description
`request`	`AISS2AO2A2M_IOCTL_AO_CLOCK_STATUS`
`arg`	`s32*`

The value returned will be one of the following.

Value	Description
`AISS2AO2A2M_AO_CLOCK_STATUS_BUSY`	The board is not ready for Analog Output clocking.
`AISS2AO2A2M_AO_CLOCK_STATUS_READY`	The board is ready for Analog Output clocking.

4.7.38. AISS2AO2A2M_IOCTL_AO_ENABLE

This service enables or disables Analog Output sampling.

Usage

Argument	Description
`request`	`AISS2AO2A2M_IOCTL_AO_ENABLE`

`arg`	`s32*`

Valid argument values are as follows.

Value	Description
`-1`	Retrieve the current setting.
`AISS2AO2A2M_AO_ENABLE_NO`	This disables Analog Output sampling.
`AISS2AO2A2M_AO_ENABLE_YES`	This enables Analog Output samplings.

4.7.39. AISS2AO2A2M_IOCTL_AO_IO_ABORT

This service aborts an ongoing `aiss2ao2a2m_write_ao()` request.

Usage

Argument	Description
`request`	`AISS2AO2A2M_IOCTL_AO_IO_ABORT`
`arg`	`s32*`

The results are reported as one of the following values.

Value	Description
`AISS2AO2A2M_AO_IO_ABORT_NO`	A write request was not aborted as none were ongoing.
`AISS2AO2A2M_AO_IO_ABORT_YES`	A write request was aborted.

4.7.40. AISS2AO2A2M_IOCTL_AO_IO_MODE

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

Usage

Argument	Description
`request`	`AISS2AO2A2M_IOCTL_AO_IO_MODE`
`arg`	`s32*`

Valid argument values are as follows. See also section 8.4 on page 63.

Value	Description
`-1`	Retrieve the current setting.
`GSC_IO_MODE_BMDMA`	Use Block Mode DMA.
`GSC_IO_MODE_DMDMA`	Use Demand Mode DMA.
`GSC_IO_MODE_PIO`	Use PIO. This is the default.

4.7.41. AISS2AO2A2M_IOCTL_AO_IO_OVERFLOW

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

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

Usage

Argument	Description
`request`	`AISS2AO2A2M_IOCTL_AO_IO_OVERFLOW`
`arg`	`s32*`

Valid argument values are as follows.

Value	Description
`-1`	Retrieve the current setting.
`AISS2AO2A2M_AO_IO_OVERFLOW_IGNORE`	Perform the check. This is the default.
`AISS2AO2A2M_AO_IO_OVERFLOW_CHECK`	Do not perform the check.

4.7.42. AISS2AO2A2M_IOCTL_AO_IO_TIMEOUT

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

Usage

Argument	Description
`request`	`AISS2AO2A2M_IOCTL_AO_IO_TIMEOUT`
`arg`	`s32*`

Valid argument values are as follows.

Value	Description
`-1`	Retrieve the current setting.
`0`	Do not sleep to wait for more space.
`1` to `3600`	Timeout after this number of seconds.
`AISS2AO2A2M_AO_IO_TIMEOUT_INFINITE`	Wait indefinitely.

4.7.43. AISS2AO2A2M_IOCTL_AO_OUTPUT_MODE

This service configures the board's Analog Output Mode when operating in *FIFO* mode (see `AISS2AO2A2M_IOCTL_AO_ACCESS_MODE`, section 4.7.22, page 30).

Usage

Argument	Description
`request`	`AISS2AO2A2M_IOCTL_AO_OUTPUT_MODE`
`arg`	`s32*`

Valid argument values are as follows.

Value	Description
`-1`	Retrieve the current setting.
`AISS2AO2A2M_AO_OUTPUT_MODE_SEQ`	Each channel is updated sequentially.
`AISS2AO2A2M_AO_OUTPUT_MODE_SIMUL`	All channels are updated simultaneously.

4.7.44. AISS2AO2A2M_IOCTL_AO_RANGE

This service configures the Analog Output voltage range.

Usage

Argument	**Description**
`request`	`AISS2AO2A2M_IOCTL_AO_RANGE`
`arg`	`s32*`

Valid argument values are as follows.

Value	**Description**
`-1`	Retrieve the current setting.
`AISS2AO2A2M_AO_RANGE_2_5V`	Set the Analog Output voltage range to ±2.5 volts.
`AISS2AO2A2M_AO_RANGE_5V`	Set the Analog Output voltage range to ±5 volts.
`AISS2AO2A2M_AO_RANGE_10V`	Set the Analog Output voltage range to ±10 volts.

4.7.45. AISS2AO2A2M_IOCTL_AO_SW_CLOCK

This service initiates a manual clock cycle for Analog Output sampling. The driver returns immediately and does not wait for completion.

Usage

Argument	**Description**
`request`	`AISS2AO2A2M_IOCTL_AO_SW_CLOCK`
`arg`	Not used.

4.7.46. AISS2AO2A2M_IOCTL_AO_SW_TRIGGER

This service initiates a manual trigger cycle for Analog Output bursting. The driver returns immediately and does not wait for completion.

Usage

Argument	**Description**
`request`	`AISS2AO2A2M_IOCTL_AO_SW_TRIGGER`
`arg`	Not used.

4.7.47. AISS2AO2A2M_IOCTL_AO_THRESH_LVL

This service configures the Analog Output buffer threshold level.

Usage

Argument	**Description**
`request`	`AISS2AO2A2M_IOCTL_AO_THRESH_LVL`
`arg`	`s32*`

Valid argument values are from zero to `0x3FFFFF`, and `-1`. A value of `-1` will return the current threshold level setting.

4.7.48. AISS2AO2A2M_IOCTL_AO_THRESH_STS

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

Usage

Argument	Description
`request`	`AISS2AO2A2M_IOCTL_AO_THRESH_STS`
`arg`	`s32*`

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

Value	Description
`AISS2AO2A2M_AO_THRESH_STS_CLEAR`	The buffer contains Threshold Level number of data values, or fewer.
`AISS2AO2A2M_AO_THRESH_STS_SET`	The buffer contains more than Threshold Level number of data values.

4.7.49. AISS2AO2A2M_IOCTL_AUTOCAL_AI

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

Usage

Argument	Description
`request`	`AISS2AO2A2M_IOCTL_AUTOCAL_AI`
`arg`	Not used.

4.7.50. AISS2AO2A2M_IOCTL_AUTOCAL_ALL

This service initiates an auto-calibration cycle of all Analog Input and Analog Output channels. Most configuration settings should be made before running an auto-calibration cycle. The driver waits for the operation to complete before returning.

Usage

Argument	Description
`request`	`AISS2AO2A2M_IOCTL_AUTOCAL_ALL`
`arg`	Not used.

4.7.51. AISS2AO2A2M_IOCTL_AUTOCAL_STS

This service returns the status of the most recent auto-calibration cycle.

Usage

Argument	Description
`request`	`AISS2AO2A2M_IOCTL_AUTOCAL_STS`
`arg`	`s32*`

Valid argument values are as follows.

Value	Description
`AISS2AO2A2M_AUTOCAL_STS_ACTIVE`	The auto-calibration cycle is still in progress.
`AISS2AO2A2M_AUTOCAL_STS_FAIL`	The auto-calibration cycle failed.
`AISS2AO2A2M_AUTOCAL_STS_PASS`	The auto-calibration cycle passed.

4.7.52. AISS2AO2A2M_IOCTL_BDO_ACCESS_MODE

This service configures the Buffered Digital Output access mode.

Usage

Argument	Description
`request`	`AISS2AO2A2M_IOCTL_BDO_ACCESS_MODE`
`arg`	`s32*`

Valid argument values are as follows.

Value	Description
`-1`	Retrieve the current setting.
`AISS2AO2A2M_BDO_ACCESS_MODE_FIFO`	Output is generated via the FIFO, which is access via the `aiss2ao2a2m_write_bdo()` API call.
`AISS2AO2A2M_BDO_ACCESS_MODE_REG`	Output is generated via direct register writes.

4.7.53. AISS2AO2A2M_IOCTL_BDO_BUF_CLEAR

This service immediately clears the current content from the Buffered Digital Output buffer. This service does not halt output clocking.

Usage

Argument	Description
`request`	`AISS2AO2A2M_IOCTL_BDO_BUF_CLEAR`
`arg`	Not used.

4.7.54. AISS2AO2A2M_IOCTL_BDO_BUF_ENABLE

This service enables or disables write access to the Buffered Digital Output buffer.

Usage

Argument	Description
`request`	`AISS2AO2A2M_IOCTL_BDO_BUF_ENABLE`
`arg`	`s32*`

Valid argument values are as follows.

Value	Description
`-1`	This option retrieves the current setting.
`AISS2AO2A2M_BDO_BUF_ENABLE_NO`	This option disables buffer access.
`AISS2AO2A2M_BDO_BUF_ENABLE_YES`	This option enables buffer access.

4.7.55. AISS2AO2A2M_IOCTL_BDO_BUF_LEVEL

This service returns the current number of 8-bit data items in the Buffered Digital Output buffer.

Usage

Argument	Description
`request`	`AISS2AO2A2M_IOCTL_BDO_BUF_LEVEL`

`arg`	`s32*`

The value returned will be from zero to 256K (262,144).

4.7.56. AISS2AO2A2M_IOCTL_BDO_BUF_OVERFLOW

This service operates on the Buffered Digital Output overflow status.

Usage

Argument	Description
`request`	`AISS2AO2A2M_IOCTL_BDO_BUF_OVERFLOW`
`arg`	`s32*`

Valid argument values supplied to the service are as follows.

Value	Description
`-1`	Retrieve the current state.
`AISS2AO2A2M_BDO_BUF_OVERFLOW_CHECK`	Check the overflow status.
`AISS2AO2A2M_BDO_BUF_OVERFLOW_CLEAR`	Clear the overflow status.

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

Value	Description
`AISS2AO2A2M_BDO_BUF_OVERFLOW_NO`	The buffer has not experienced an overflow condition.
`AISS2AO2A2M_BDO_BUF_OVERFLOW_YES`	The buffer has experienced an overflow condition.

4.7.57. AISS2AO2A2M_IOCTL_BDO_BUF_UNDERFLOW

This service operates on the Buffered Digital Output underflow status.

Usage

Argument	Description
`request`	`AISS2AO2A2M_IOCTL_BDO_BUF_UNDERFLOW`
`arg`	`s32*`

Valid argument values supplied to the service are as follows.

Value	Description
`-1`	Retrieve the current state.
`AISS2AO2A2M_BDO_BUF_UNDERFLOW_CHECK`	Check the underflow status.
`AISS2AO2A2M_BDO_BUF_UNDERFLOW_CLEAR`	Clear the underflow status.

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

Value	Description
`AISS2AO2A2M_BDO_BUF_UNDERFLOW_NO`	The buffer has not experienced an underflow condition.
`AISS2AO2A2M_BDO_BUF_UNDERFLOW_YES`	The buffer has experienced an underflow condition.

4.7.58. AISS2AO2A2M_IOCTL_BDO_CLOCK_ENABLE

This service enables or disables clocking of the Buffered Digital Output.

Usage

Argument	Description
`request`	`AISS2AO2A2M_IOCTL_BDO_CLOCK_ENABLE`
`arg`	`s32*`

Valid argument values are as follows.

Value	Description
`-1`	This option retrieves the current setting.
`AISS2AO2A2M_BDO_CLOCK_ENABLE_NO`	This option disables clocking.
`AISS2AO2A2M_BDO_CLOCK_ENABLE_YES`	This option enables clocking.

4.7.59. AISS2AO2A2M_IOCTL_BDO_CLOCK_SOURCE

This service configures the Buffered Digital Output clock source selection.

Usage

Argument	Description
`request`	`AISS2AO2A2M_IOCTL_BDO_CLOCK_SOURCE`
`arg`	`s32*`

Valid argument values are as follows.

Value	Description
`-1`	Retrieve the current setting.
`AISS2AO2A2M_BDO_CLOCK_SOURCE_EXT`	This selects external clocking.
`AISS2AO2A2M_BDO_CLOCK_SOURCE_INT`	This selects internal clocking from the BDO Rate Generator.

4.7.60. AISS2AO2A2M_IOCTL_BDO_IO_ABORT

This service aborts an ongoing Buffered Digital Output write request.

Usage

Argument	Description
`request`	`AISS2AO2A2M_IOCTL_BDO_IO_ABORT`
`arg`	`s32*`

The results are reported as one of the following values.

Value	Description
`AISS2AO2A2M_BDO_IO_ABORT_NO`	A write request was not aborted as none were ongoing.
`AISS2AO2A2M_BDO_IO_ABORT_YES`	An ongoing write request was aborted.

4.7.61. AISS2AO2A2M_IOCTL_BDO_IO_MODE

This service sets the I/O mode used for Buffered Digital Output write requests.

Usage

Argument	**Description**
`request`	`AISS2AO2A2M_IOCTL_BDO_IO_MODE`
`arg`	`s32*`

Valid argument values are as follows. See also section 8.4 on page 63.

Value	**Description**
`-1`	Retrieve the current setting.
`GSC_IO_MODE_PIO`	Use PIO mode. This is the default.

4.7.62. AISS2AO2A2M_IOCTL_BDO_IO_OVERFLOW

This service configures the Buffered Digital Output write service check for output buffer overflows. Output data is lost when there is an overflow. If the check is performed and an overflow is detected, then the write service immediately returns an error.

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

Usage

Argument	**Description**
`request`	`AISS2AO2A2M_IOCTL_BDO_IO_OVERFLOW`
`arg`	`s32*`

Valid argument values are as follows.

Value	**Description**
`-1`	Retrieve the current setting.
`AISS2AO2A2M_BDO_IO_OVERFLOW_CHECK`	Perform the check. This is the default.
`AISS2AO2A2M_BDO_IO_OVERFLOW_IGNORE`	Do not perform the check.

4.7.63. AISS2AO2A2M_IOCTL_BDO_IO_TIMEOUT

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

Usage

Argument	**Description**
`request`	`AISS2AO2A2M_IOCTL_BDO_IO_TIMEOUT`
`arg`	`s32*`

Valid argument values are as follows.

Value	**Description**
`-1`	Retrieve the current setting.
`0`	Do not sleep to wait for more space.
`1` to `3600`	Timeout after this number of seconds.
`AISS2AO2A2M_BDO_IO_TIMEOUT_INFINITE`	Wait indefinitely.

4.7.64. AISS2AO2A2M_IOCTL_BDO_IO_UNDERFLOW

This service configures the Buffered Digital Output write service check for output buffer underflows. Indeterminate data is returned when there is an underflow. If the check is performed and an underflow is detected, then the write service immediately returns an error.

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

Usage

Argument	Description
`request`	`AISS2AO2A2M_IOCTL_BDO_IO_UNDERFLOW`
`arg`	`s32*`

Valid argument values are as follows.

Value	Description
`-1`	Retrieve the current setting.
`AISS2AO2A2M_BDO_IO_UNDERFLOW_CHECK`	Perform the check. This is the default.
`AISS2AO2A2M_BDO_IO_UNDERFLOW_IGNORE`	Do not perform the check.

4.7.65. AISS2AO2A2M_IOCTL_BDO_RATE_GEN_ENABLE

This service enables or disables the Buffered Digital Output Rate Generator, which is used for output clocking.

Usage

Argument	Description
`request`	`AISS2AO2A2M_IOCTL_BDO_RATE_GEN_ENABLE`
`arg`	`s32*`

Valid argument values are as follows.

Value	Description
`-1`	Retrieve the current setting.
`AISS2AO2A2M_BDO_RATE_GEN_ENABLE_NO`	This option disables the rate generator.
`AISS2AO2A2M_BDO_RATE_GEN_ENABLE_YES`	This option enables the rate generator.

4.7.66. AISS2AO2A2M_IOCTL_BDO_RATE_GEN_NDIV

This service sets the NDIV divider value for the Buffered Digital Output Rate Generator.

Usage

Argument	Description
`request`	`AISS2AO2A2M_IOCTL_BDO_RATE_GEN_NDIV`
`arg`	`s32*`

Valid argument values are in the range from eight to `0xFFFFFF`, and `-1`. The value `-1` is used to retrieve the current setting. The minimum value varies according to the board's master clock so that the maximum rate generator output is approximately 5MHz.

4.7.67. AISS2AO2A2M_IOCTL_BDO_SW_CLOCK

This service initiates a manual clock cycle for the Buffered Digital Output. The driver returns immediately and does not wait for completion.

Usage

Argument	Description
`request`	`AISS2AO2A2M_IOCTL_BDO_SW_CLOCK`
`arg`	Not used.

4.7.68. AISS2AO2A2M_IOCTL_BDO_THRESH_LVL

This service configures the Buffered Digital Output buffer threshold level.

Usage

Argument	Description
`request`	`AISS2AO2A2M_IOCTL_BDO_THRESH_LVL`
`arg`	`s32*`

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

4.7.69. AISS2AO2A2M_IOCTL_BDO_THRESH_STS

This service retrieves the current threshold level status for the Buffered Digital Output buffer, which indicates whether or not there is more than Threshold Level number of 8-bit data items in the buffer.

Usage

Argument	Description
`request`	`AISS2AO2A2M_IOCTL_BDO_THRESH_STS`
`arg`	`s32*`

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

Value	Description
`AISS2AO2A2M_BDO_THRESH_STS_CLEAR`	The buffer contains Threshold Level number of data values, or fewer.
`AISS2AO2A2M_BDO_THRESH_STS_SET`	The buffer contains more than Threshold Level number of data values.

4.7.70. AISS2AO2A2M_IOCTL_CBL_IN_CLK_IO

This service configures the Cable Input Clock I/O pin operation.

Usage

Argument	Description
`request`	`AISS2AO2A2M_IOCTL_CBL_IN_CLK_IO`
`arg`	`s32*`

Valid argument values are as follows.

Value	Description
`-1`	Retrieve the current setting.
`AISS2AO2A2M_CBL_IN_CLK_IO_FUNC`	The pin performs as programmed.
`AISS2AO2A2M_CBL_IN_CLK_IO_OUT_0`	The pin outputs a logic low level.

4.7.71. AISS2AO2A2M_IOCTL_CBL_IO_CLK_DIR

This service configures the Cable I/O Clock pin direction.

Usage

Argument	Description
`request`	`AISS2AO2A2M_IOCTL_CBL_IO_CLK_DIR`
`arg`	`s32*`

Valid argument values are as follows.

Value	Description
`-1`	Retrieve the current setting.
`AISS2AO2A2M_CBL_IO_CLK_DIR_IN`	The pin operates as an input.
`AISS2AO2A2M_CBL_IO_CLK_DIR_OUT`	The pin operates as an output.

4.7.72. AISS2AO2A2M_IOCTL_CBL_OUT_CLK_IO

This service configures the Cable Output Clock I/O pin operation.

Usage

Argument	Description
`request`	`AISS2AO2A2M_IOCTL_CBL_OUT_CLK_IO`
`arg`	`s32*`

Valid argument values are as follows.

Value	Description
`-1`	Retrieve the current setting.
`AISS2AO2A2M_CBL_OUT_CLK_IO_FUNC`	The pins perform as programmed.
`AISS2AO2A2M_CBL_OUT_CLK_IO_OUT_0`	The pins output a logic low level.

4.7.73. AISS2AO2A2M_IOCTL_CBL_TRIG_IO

This service configures the Cable Trigger I/O pin operation.

Usage

Argument	Description
`request`	`AISS2AO2A2M_IOCTL_CBL_TRIG_IO`
`arg`	`s32*`

Valid argument values are as follows.

Value	Description
`-1`	Retrieve the current setting.
`AISS2AO2A2M_CBL_TRIG_IO_FUNC`	The pins perform as programmed.

`AISS2AO2A2M_CBL_TRIG_IO_OUT_0`	The pins output a logic low level.

4.7.74. AISS2AO2A2M_IOCTL_DATA_FORMAT

This service sets the data encoding format.

Usage

Argument	**Description**
`request`	`AISS2AO2A2M_IOCTL_DATA_FORMAT`
`arg`	`s32*`

Valid argument values are as follows.

Value	**Description**
`-1`	Retrieve the current setting.
`AISS2AO2A2M_DATA_FORMAT_2S_COMP`	Select the Twos Compliment data format.
`AISS2AO2A2M_DATA_FORMAT_OFF_BIN`	Select the Offset Binary encoding format.

4.7.75. AISS2AO2A2M_IOCTL_DIO_DIR

This service configures the direction of the 8-bit bidirectional digital I/O port.

Usage

Argument	**Description**
`request`	`AISS2AO2A2M_IOCTL_DIO_DIR`
`arg`	`s32*`

Valid argument values are as follows.

Value	**Description**
`-1`	Retrieve the current setting.
`AISS2AO2A2M_DIO_DIR_IN`	This selects the input configuration.
`AISS2AO2A2M_DIO_DIR_OUT`	This selects the output configuration.

4.7.76. AISS2AO2A2M_IOCTL_DIO_READ

This service retrieves the signal levels for the eight bidirectional digital I/O pins.

Usage

Argument	**Description**
`request`	`AISS2AO2A2M_IOCTL_DIO_READ`
`arg`	`s32*`

Argument values returned are from zero to `0xFF`.

4.7.77. AISS2AO2A2M_IOCTL_DIO_WRITE

This service applies values to the digital I/O cable signals. The lower eight bits are applied to the 8-bit bidirectional digital I/O port. If this port is configured as an output, then the value appears at the cable interface. The next eight bits are applied to the 8-bit Buffered Digital Output port, if it is configured for register based access. If it is configured for FIFO access, then these eight bits are ignored. The upper sixteen bits are ignored.

Usage

Argument	**Description**
`request`	`AISS2AO2A2M_IOCTL_DIO_WRITE`
`arg`	`s32*`

Valid argument values are from zero to `0xFFFF`.

4.7.78. AISS2AO2A2M_IOCTL_INITIALIZE

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

Usage

Argument	**Description**
`request`	`AISS2AO2A2M_IOCTL_INITIALIZE`
`arg`	Not used.

4.7.79. AISS2AO2A2M_IOCTL_IRQ_ENABLE

This service enables and disables device interrupts. If a bit is set, then the interrupt is enabled. If a bit is clear, then the interrupt is disabled. Interrupts remain enabled until disabled by the application.

Usage

Argument	**Description**
`request`	`AISS2AO2A2M_IOCTL_IRQ_ENABLE`
`arg`	`s32*`

Valid argument values include any bitwise combination of the following bits.

Value	**Description**
`-1`	Retrieve the current setting.
`AISS2AO2A2M_IRQ_AI_BURST_DONE`	This refers to the completion of an Analog Input burst.
`AISS2AO2A2M_IRQ_AI_BURST_START`	This refers to the start of an Analog Input burst.
`AISS2AO2A2M_IRQ_AI_FAULT`	This refers to an Analog Input buffer overflow.
`AISS2AO2A2M_IRQ_AI_THRESH_H2L`	This refers to the Analog Input buffer fill level dropping to the threshold level or below.
`AISS2AO2A2M_IRQ_AI_THRESH_L2H`	This refers to the Analog Input buffer fill level rising to exceed the threshold level.
`AISS2AO2A2M_IRQ_AO_BURST_READY`	This refers to the Analog Output buffer Burst Ready status going from the *busy* to the *ready* state.
`AISS2AO2A2M_IRQ_AO_FAULT`	This refers to an Analog Output buffer overflow or underflow.
`AISS2AO2A2M_IRQ_AO_LOAD_RDY_H2L`	This refers to the Analog Output buffer Load Ready status going from the high to the low state.
`AISS2AO2A2M_IRQ_AO_LOAD_RDY_L2H`	This refers to the Analog Output buffer Load Ready status going from the low to the high state.
`AISS2AO2A2M_IRQ_AO_THRESH_H2L`	This refers to the Analog Output buffer fill level dropping to the threshold level or below.
`AISS2AO2A2M_IRQ_AO_THRESH_L2H`	This refers to the Analog Output buffer fill level rising to exceed the threshold level.
`AISS2AO2A2M_IRQ_AUTO_CAL_DONE`	This refers to completion of an auto-calibration cycle.

AISS2AO2A2M_IRQ_BDO_THRESH_H2L	This refers to the BDO buffer fill level dropping to the threshold level or below.
AISS2AO2A2M_IRQ_BDO_THRESH_L2H	This refers to the BDO buffer fill level rising to exceed the threshold level.
AISS2AO2A2M_IRQ_DIO_0_L2H	This refers to one a appearing at the cable's bidirectional digital signal zero.

4.7.80. AISS2AO2A2M_IOCTL_QUERY

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

Usage

Argument	Description
request	AISS2AO2A2M_IOCTL_QUERY
arg	s32*

Valid argument values are as follows.

Value	Description
AISS2AO2A2M_QUERY_AI_CHANS_MAX	This returns the maximum number of Analog Input channels supported by the board, which may be more that the board's current configuration.
AISS2AO2A2M_QUERY_AI_CHANS_QTY	This returns the actual number of Analog Input channels on the current board.
AISS2AO2A2M_QUERY_AI_FIFO_SIZE	This returns the size of the Analog Input buffer in 32-bit values.
AISS2AO2A2M_QUERY_AI_FSAMP_MAX	This gives the maximum Analog Input FSAMP rate.
AISS2AO2A2M_QUERY_AI_FSAMP_MIN	This gives the minimum Analog Input FSAMP rate.
AISS2AO2A2M_QUERY_AI_NDIV_MAX	This gives the maximum Analog Input NDIV value.
AISS2AO2A2M_QUERY_AI_NDIV_MIN	This gives the minimum Analog Input NDIV value.
AISS2AO2A2M_QUERY_AO_CHANS_MAX	This returns the maximum number of Analog Output channels supported by the board, which may be more that the board's current configuration.
AISS2AO2A2M_QUERY_AO_CHANS_QTY	This returns the actual number of Analog Output channels on the current board.
AISS2AO2A2M_QUERY_AO_FIFO_SIZE	This returns the size of the Analog Output buffer in 32-bit values.
AISS2AO2A2M_QUERY_AO_FSAMP_MAX	This gives the maximum Analog Output FSAMP rate.
AISS2AO2A2M_QUERY_AO_FSAMP_MIN	This gives the minimum Analog Output FSAMP rate.
AISS2AO2A2M_QUERY_AO_NDIV_MAX	This gives the maximum Analog Output NDIV value.
AISS2AO2A2M_QUERY_AO_NDIV_MIN	This gives the minimum Analog Output NDIV value.
AISS2AO2A2M_QUERY_AUTOCAL_MS	This returns the maximum duration of the Auto Calibration cycle in milliseconds. This pertains to both the AI and the *all* auto-calibration options.
AISS2AO2A2M_QUERY_BDO_FIFO_SIZE	This returns the size of the BDO buffer in 8-bit values.
AISS2AO2A2M_QUERY_BDO_FSAMP_MAX	This gives the maximum BDO FSAMP rate.
AISS2AO2A2M_QUERY_BDO_FSAMP_MIN	This gives the minimum BDO FSAMP rate.
AISS2AO2A2M_QUERY_BDO_NDIV_MAX	This gives the maximum BDO NDIV value.
AISS2AO2A2M_QUERY_BDO_NDIV_MIN	This gives the minimum BDO NDIV value.
AISS2AO2A2M_QUERY_COUNT	This returns the number of query options supported by the IOCTL service.
AISS2AO2A2M_QUERY_DEVICE_TYPE	This returns the identifier value for the board's type. This should be `GSC_DEV_TYPE_16AISS2AO2A2M`.

`AISS2AO2A2M_QUERY_INITIALIZE_MS`	This returns the duration of a board initialization in milliseconds.
`AISS2AO2A2M_QUERY_MASTER_CLOCK`	This returns the master clock frequency in hertz.
`AISS2AO2A2M_QUERY_RATE_GEN_QTY`	This returns the number of board Rate Generators.

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

Value	Description
`AISS2AO2A2M_IOCTL_QUERY_ERROR`	Either there was a processing error or the query option is unrecognized.

4.7.81. AISS2AO2A2M_IOCTL_RATE_A_GEN_ENABLE

This service enables or disables the Rate-A Generator, which is used for Analog Input sampling.

Usage

Argument	Description
`request`	`AISS2AO2A2M_IOCTL_RATE_A_GEN_ENABLE`
`arg`	`s32*`

Valid argument values are as follows.

Value	Description
`-1`	Retrieve the current setting.
`AISS2AO2A2M_RATE_A_GEN_ENABLE_NO`	This option disables the rate generator.
`AISS2AO2A2M_RATE_A_GEN_ENABLE_YES`	This option enables the rate generator.

4.7.82. AISS2AO2A2M_IOCTL_RATE_A_GEN_NDIV

This service sets the NDIV divider value for the Rate-A Generator.

Usage

Argument	Description
`request`	`AISS2AO2A2M_IOCTL_RATE_A_GEN_NDIV`
`arg`	`s32*`

Valid argument values are in the range from `20` to `0xFFFFFF`, and `-1`. The value `-1` is used to retrieve the current setting.

4.7.83. AISS2AO2A2M_IOCTL_RATE_B_GEN_ENABLE

This service enables or disables the Rate-B Generator, which is used for Analog Input and/or Analog Output bursting.

Usage

Argument	Description
`request`	`AISS2AO2A2M_IOCTL_RATE_B_GEN_ENABLE`
`arg`	`s32*`

Valid argument values are as follows.

Value	Description
`-1`	Retrieve the current setting.
`AISS2AO2A2M_RATE_B_GEN_ENABLE_NO`	This option disables the rate generator.
`AISS2AO2A2M_RATE_B_GEN_ENABLE_YES`	This option enables the rate generator.

4.7.84. AISS2AO2A2M_IOCTL_RATE_B_GEN_NDIV

This service sets the NDIV divider value for the Rate-B Generator.

Usage

Argument	Description
`request`	`AISS2AO2A2M_IOCTL_RATE_B_GEN_NDIV`
`arg`	`s32*`

Valid argument values are in the range from `20` to `0xFFFFFF`, and `-1`. The value `-1` is used to retrieve the current setting.

4.7.85. AISS2AO2A2M_IOCTL_RATE_C_GEN_ENABLE

This service enables or disables the Rate-C Generator, which is used for Analog Output sampling.

Usage

Argument	Description
`request`	`AISS2AO2A2M_IOCTL_RATE_C_GEN_ENABLE`
`arg`	`s32*`

Valid argument values are as follows.

Value	Description
`-1`	Retrieve the current setting.
`AISS2AO2A2M_RATE_C_GEN_ENABLE_NO`	This option disables the rate generator.
`AISS2AO2A2M_RATE_C_GEN_ENABLE_YES`	This option enables the rate generator.

4.7.86. AISS2AO2A2M_IOCTL_RATE_C_GEN_NDIV

This service sets the NDIV divider value for the Rate-C Generator.

Usage

Argument	Description
`request`	`AISS2AO2A2M_IOCTL_RATE_C_GEN_NDIV`
`arg`	`s32*`

Valid argument values are in the range from `20` to `0xFFFFFF`, and `-1`. The value `-1` is used to retrieve the current setting.

4.7.87. AISS2AO2A2M_IOCTL_REG_MOD

This service performs a read-modify-write of a 16AISS2AO2A2M register. This includes only the GSC firmware registers. The PCI and PLX Feature Set Registers are read-only. Refer to `16aiss2ao2a2m.h` for the complete list of GSC firmware registers.

Usage

Argument	Description
`request`	`AISS2AO2A2M_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.

4.7.88. AISS2AO2A2M_IOCTL_REG_READ

This service reads the value of a 16AISS2AO2A2M register. This includes the PCI registers, the PLX Feature Set Registers and the GSC firmware registers. Refer to `16aiss2ao2a2m.h` and `gsc_pci9056.h` for the complete list of accessible registers.

Usage

Argument	Description
`request`	`AISS2AO2A2M_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.

4.7.89. AISS2AO2A2M_IOCTL_REG_WRITE

This service writes a value to a 16AISS2AO2A2M register. This includes only the GSC firmware registers. The PCI and PLX Feature Set Registers are read-only. Refer to `16aiss2ao2a2m.h` for a complete list of the GSC firmware registers.

Usage

Argument	**Description**
`request`	`AISS2AO2A2M_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.

4.7.90. AISS2AO2A2M_IOCTL_TRIGGER_DIR

This service configures the direction of the cable trigger signal.

Usage

Argument	**Description**
`request`	`AISS2AO2A2M_IOCTL_TRIGGER_DIR`
`arg`	`s32*`

Valid argument values are as follows.

Value	**Description**
`-1`	Retrieve the current setting.
`AISS2AO2A2M_TRIGGER_DIR_IN`	The signal is an input.
`AISS2AO2A2M_TRIGGER_DIR_OUT`	The signal is an output.

4.7.91. AISS2AO2A2M_IOCTL_WAIT_CANCEL

This service resumes all threads blocked via `AISS2AO2A2M_IOCTL_WAIT_EVENT` IOCTL calls (section 4.7.92, page 54), 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`	`AISS2AO2A2M_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 4.7.92.2 on page 55.
`gsc`	This specifies the set of `AISS2AO2A2M_WAIT_GSC_*` events whose wait requests are to be cancelled. Refer to section 4.7.92.3 on page 55.
`alt`	This is unused by the 16AISS2AO2A2M driver and should be zero.
`io`	This specifies the set of `AISS2AO2A2M_WAIT_IO_*` events whose wait requests are to be cancelled. Refer to section 4.7.92.4 on page 56.
`timeout_ms`	This is unused by wait cancel operations.
`count`	Upon return this indicates the number of waits that were cancelled.

4.7.92. AISS2AO2A2M_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:** 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`	`AISS2AO2A2M_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 4.7.92.1on page 55.
`main`	This specifies any number of `GSC_WAIT_MAIN_*` events that the thread is to wait for. Refer to section 4.7.92.2 on page 55.
`gsc`	This specifies any number of `AISS2AO2A2M_WAIT_GSC_*` events that the thread is to wait for. Refer to section 4.7.92.3 on page 55.
`alt`	This is unused by the 16AISS2AO2A2M driver and must be zero.
`io`	This specifies any number of `AISS2AO2A2M_WAIT_IO_*` events that the thread is to wait for. Refer to section 4.7.92.4 on page 56.
`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.

4.7.92.1. `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 option 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.

4.7.92.2. `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 16AISS2AO2A2M 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 16AISS2AO2A2M.
`GSC_WAIT_MAIN_PCI`	This refers to any interrupt generated by the 16AISS2AO2A2M.
`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.

4.7.92.3. `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 `AISS2AO2A2M_IOCTL_IRQ_ENABLE` (section 4.7.79, page 48). The interrupts remain enabled after they occur.

Fields	Description
`AISS2AO2A2M_WAIT_GSC_AI_BURST_DONE`	This refers to the completion of an Analog Input burst.
`AISS2AO2A2M_WAIT_GSC_AI_BURST_START`	This refers to the start of an Analog Input burst.
`AISS2AO2A2M_WAIT_GSC_AI_FAULT`	This refers to an Analog Input buffer overflow.

`AISS2AO2A2M_WAIT_GSC_AI_THRESH_H2L`	This refers to the Analog Input buffer fill level dropping to the threshold level or below.
`AISS2AO2A2M_WAIT_GSC_AI_THRESH_L2H`	This refers to the Analog Input buffer fill level rising to exceed the threshold level.
`AISS2AO2A2M_WAIT_GSC_AO_BURST_READY`	This refers to the Analog Output Burst status going from the *busy* state to the *ready* state.
`AISS2AO2A2M_WAIT_GSC_AO_FAULT`	This refers to an Analog Output buffer overflow or underflow.
`AISS2AO2A2M_WAIT_GSC_AO_LOAD_RDY_H2L`	This refers to the Analog Output Load Ready status going from the high to the low state.
`AISS2AO2A2M_WAIT_GSC_AO_LOAD_RDY_L2H`	This refers to the Analog Output Load Ready status going from the low to the high state.
`AISS2AO2A2M_WAIT_GSC_AO_THRESH_H2L`	This refers to the Analog Output buffer fill level dropping to the threshold level or below.
`AISS2AO2A2M_WAIT_GSC_AO_THRESH_L2H`	This refers to the Analog Output buffer fill level rising to exceed the threshold level.
`AISS2AO2A2M_WAIT_GSC_AUTO_CAL_DONE`	This refers to completion of an auto-calibration cycle. This refers to both the *AI* and the *all* auto-calibration options.
`AISS2AO2A2M_WAIT_GSC_BDO_THRESH_H2L`	This refers to the BDO buffer fill level dropping to the threshold level or below.
`AISS2AO2A2M_WAIT_GSC_BDO_THRESH_L2H`	This refers to the BDO buffer fill level rising to exceed the threshold level.
`AISS2AO2A2M_WAIT_GSC_DIO_0_L2H`	This refers to a "`1`" appearing at the cable's bidirectional digital signal zero.

4.7.92.4. `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 I/O requests.

Fields	**Description**
`AISS2AO2A2M_WAIT_IO_AI_RX_ABORT`	This refers to AI read requests which have been aborted.
`AISS2AO2A2M_WAIT_IO_AI_RX_DONE`	This refers to AI read requests which have ended.
`AISS2AO2A2M_WAIT_IO_AI_RX_ERROR`	This refers to AI read requests which end due to an error.
`AISS2AO2A2M_WAIT_IO_AI_RX_TIMEOUT`	This refers to AI read requests which end due to the timeout period lapse.
`AISS2AO2A2M_WAIT_IO_AO_TX_ABORT`	This refers to AO write requests which have been aborted.
`AISS2AO2A2M_WAIT_IO_AO_TX_DONE`	This refers to AO write requests which have ended.
`AISS2AO2A2M_WAIT_IO_AO_TX_ERROR`	This refers to AO write requests which end due to an error.
`AISS2AO2A2M_WAIT_IO_AO_TX_TIMEOUT`	This refers to AO write requests which end due to the timeout period lapse.
`AISS2AO2A2M_WAIT_IO_BDO_TX_ABORT`	This refers to BDO write requests which have been aborted.
`AISS2AO2A2M_WAIT_IO_BDO_TX_DONE`	This refers to BDO write requests which have ended.
`AISS2AO2A2M_WAIT_IO_BDO_TX_ERROR`	This refers to BDO write requests which end due to an error.
`AISS2AO2A2M_WAIT_IO_BDO_TX_TIMEOUT`	This refers to BDO write requests which end due to the timeout period lapse.

4.7.93. AISS2AO2A2M_IOCTL_WAIT_STATUS

This service counts the number of threads blocked via `AISS2AO2A2M_IOCTL_WAIT_EVENT` IOCTL calls (section 4.7.92, page 54), 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`	`AISS2AO2A2M_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 4.7.92.2 on page 55.
`gsc`	This specifies the set of `AISS2AO2A2M_WAIT_GSC_*` events whose wait requests are to be counted. Refer to section 4.7.92.3 on page 55.
`alt`	This is unused by the 16AISS2AO2A2M driver and should be zero.
`io`	This specifies the set of `AISS2AO2A2M_WAIT_IO_*` events whose wait requests are to be counted. Refer to section 4.7.92.4 on page 56.
`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 source files are summarized in the table below.

File	**Description**
`driver/*.c`	These are the driver source files.
`driver/*.h`	These are the driver header files.
`driver/16aiss2ao2a2m.h`	A driver interface header file.
`driver/Makefile`	This is the driver make file.
`driver/start`	Shell script to install the driver executable and device nodes.

5.2. Build

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

Follow the below steps to build the driver.

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

2. Remove existing build targets using the below command line.

```
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 insure that the driver module in the install directory is the module that is loaded. This is accomplished by making sure that an already loaded module is first unloaded before attempting to load the module from the disk drive. In addition, the script also deletes and recreates the device nodes. This is done to insure that the device nodes in use have the same major number as assigned dynamically to the driver by the kernel, and so that the number of device nodes correspond to the number of boards identified by the driver.

5.3.1. Manual Driver Startup Procedures

Start the driver manually by following the below listed steps.

NOTE: The following steps may require elevated privileges.

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

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

   ```
   ./start
   ```

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

   NOTE: The 16AISS2AO2A2M 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 `16aiss2ao2a2m` 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/16aiss2ao2a2m.*
   ```

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/16aiss2ao2a2m/driver/start
   ```

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.

5.4. Verification

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

1. Verify that the file `/proc/16aiss2ao2a2m` 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/16aiss2ao2a2m
   ```

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. This should be viewable via the text file `/var/log/messages`, or from the `dmesg` and/or `journelctl` commands. It is also reported in the text file `/proc/16aiss2ao2a2m` while the driver is loaded and running.

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 16aiss2ao2a2m
   ```

2. Verify that the driver module has been unloaded by issuing the below command. The module name `16aiss2ao2a2m` should not be in the listed output.

   ```
   lsmod
   ```

6. Document Source Code Examples

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

6.1. Files

The library files are summarized in the table below.

File	**Description**
`…/docsrc/*.c`	These are the C source files.
`…/docsrc/makefile`	This is the library make file.
`…/docsrc/makefile.dep`	This is an automatically generated make dependency file.
`…/include/16aiss2ao2a2m_dsl.h`	This is the primary utility header file.
`…/lib/16aiss2ao2a2m_dsl.a`	This is the statically linkable library file.

6.2. Build

The library is built via the Overall Make Script (section 2.7, page 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 using the below command line.

   ```
   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, page 14).

6.3. Library Use

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

Description	**File**	**Location**
Header File	`16aiss2ao2a2m_dsl.h`	`…/include/`
Static Link Library	`16aiss2ao2a2m_dsl.a`	`…/lib/`

7. Utility Source Code

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

7.1. Files

The library files are summarized in the table below.

File	**Description**
`…/utils/util_*.c`	These are device specific utility source files.
`…/utils/gsc_*.c`	These are device and OS independent utility source files.
`…/utils/os_*.c`	These are OS specific utility source files.
`…/utils/makefile`	This is the library make file.
`…/utils/makefile.dep`	This is an automatically generated make dependency file.
`…/include/16aiss2ao2a2m_utils.h`	This is the primary utility header file.
`…/lib/16aiss2ao2a2m_utils.a`	This is the statically linkable library file.

7.2. Build

The library is built via the Overall Make Script (section 2.7, page 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 using the below command line.

   ```
   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, page 14).

7.3. Library Use

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

Description	**File**	**Location**
Header File	`16aiss2ao2a2m_utils.h`	`…/include/`
Static Link Libraries	`16aiss2ao2a2m_utils.a`	`…/lib/`

8. Operating Information

This section explains some basic operational procedures for using the 16AISS2AO2A2M. This is in no way intended to be a comprehensive guide. This is simply to address a very few issues relating to their use.

8.1. Analog 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.

Item	Name/File	Location
Function	`aiss2ao2a2m_config_ai()`	Source File
Source File	`util_config_ai.c`	`…/utils/`
Header File	`16aiss2ao2a2m_utils.h`	`…/include/`
Library File	`16aiss2ao2a2m_utils.a`	`…/lib/`

8.2. Analog Output Configuration

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

Item	Name/File	Location
Function	`aiss2ao2a2m_config_ao()`	Source File
Source File	`config_ao.c`	`…/utils/`
Header File	`16aiss2ao2a2m_utils.h`	`…/include/`
Library File	`16aiss2ao2a2m_utils.a`	`…/lib/`

8.3. Buffered Digital Output Configuration

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

Item	Name/File	Location
Function	`aiss2ao2a2m_config_bdo()`	Source File
Source File	`config_bdo.c`	`…/utils/`
Header File	`16aiss2ao2a2m_utils.h`	`…/include/`
Library File	`16aiss2ao2a2m_utils.a`	`…/lib/`

8.4. I/O Modes

All I/O transfer requests (AI, AO, BDO) move the requested data between the board buffers, intermediate driver buffers, and the application buffers. The data is processed in chunks no larger than the size of the corresponding driver buffer. The process used to move data between the board's buffer and the intermediate driver buffer is according to the I/O mode selection.

> **NOTE**: The 16AISS2AO2A2M has only two DMA engines, which means only two DMA based transfers can be performed simultaneously. Demand Mode DMA transfers tend to utilize the DMA engine for the duration of the transfer while Block Mode DMA transfers utilize the DMA engine in very brief spurts. If an application intends to perform three simultaneous DMA transfers (AI, AO and BDO) then at most one can be DMDMA. If two are done using DMDMA (AI and AO), then the third (BDO) will be unable to gain access to a DMA engine.

8.4.1. PIO - Programmed I/O

This is called Programmed I/O and involves repetitive register accesses. In this mode the driver will transfer data one value at a time via a register read or write. As needed, the driver will repeatedly sleep for one system time tick in order to wait for addition data or space in the corresponding board buffer. This process is repeated until the request is satisfied or the I/O timeout expires, whichever occurs first.

8.4.2. BMDMA - Block Mode DMA

For Block Mode DMA transfers, hardware onboard the 16AISS2AO2A2M is used to transfer the data without processor intervention. In this mode the driver checks for available data or space in the appropriate board buffer. Depending on the size of the I/O request, the driver may break the request into smaller transfers in order to insure data integrity. When sufficient data or space is available a DMA transfer is performed. The volume of data moved in a given transfer changes according to a number of variables. This process is repeated until the request is satisfied or the I/O timeout expires, whichever occurs first.

8.4.3. DMDMA - Demand Mode DMA

In Demand Mode DMA, data is moved between the appropriate board buffer and the corresponding intermediate driver buffer in a single DMA transfer that occurs over time as the data or space becomes available in the board buffer. The process is repeated until the transfer is completed or the I/O timeout expires, whichever occurs first.

8.5. Debugging Aids

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

8.5.1. 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/`

8.5.2. Detailed Register Dump

Among the utility services provided is a function to generate a detailed listing of the board’s registers to the console. When used, the function is typically used to verify the board’s configuration. In these cases, the function should be called just prior to the first read or write operation. When intended for sending to GSC tech support, please set the *detail* argument to `1`. The function arguments are as follows. The utility location is given in the subsequent table.

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

Description	File/Name	Location
Function	`aiss2ao2a2m_reg_list()`	Source File
Source File	`reg.c`	`…/utils/`
Header File	`aiss2ao2a2m_utils.h`	`…/include/`
Library File	`aiss2ao2a2m_utils.a`	`…/lib/`

9. Sample Applications

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

9.1. aout – Analog Ouput - …/aout/

This application writes a pattern to the Analog Output channels.

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.

9.4. fsamp - Sample Rate - …/fsamp/

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

9.5. 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.

9.6. regs - Register Access - …/regs/

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

9.7. rxrate - Receive Rate - …/rxrate/

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

9.8. 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.

9.9. txrate - Transmit Rate - …/txrate/

This application configures the board for its highest AO or BDO rate then writes either analog or digital 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**
July 1, 2019	Updated to release version 1.1.86.28.0. Updated the kernel support table. Remove BMDMA from BDO I/O mode. Minor editorial changes. Some document reorganization. Renamed API service: …`ai_read` -> …`read_ai`, …`ao_write` -> …`write_ao` and …`bdo_write` -> …`write_bdo`.
August 24, 2018	Initial release. This is a release version 1.0.80.26.0.