16HSDI4AO4

16-bit, 4 A/D channels, 4 D/A Channels, 1M S/S/Ch
16-bit Digital I/O

PMC66-16HSDI4AO4

Linux Device Driver
And API Library
User Manual

Manual Revision: July 24, 2023
Driver Release Version 2.8.104.47.0

General Standards Corporation
8302A Whitesburg Drive
Huntsville, AL 35802
Phone: (256) 880-8787
Fax: (256) 880-8788
URL: http://www.generalstandards.com
E-mail: sales@generalstandards.com
E-mail: support@generalstandards.com

Preface

Additional copies of this manual or other literature may be obtained from:

General Standards Corporation
8302A Whitesburg Dr.
Huntsville, Alabama 35802
Phone: (256) 880-8787
FAX: (256) 880-8788
URL: http://www.generalstandards.com
E-mail: sales@generalstandards.com

Table of Contents

Table of Figures

1. Introduction

1.1. Purpose

The purpose of this document is to describe the interface to the 16HSDI4AO4 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 16HSDI4AO4 hardware. The API Library and driver interfaces are based on the board's functionality.

1.2. Acronyms

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

Acronyms	Description
ADC	Analog-to-Digital Converter
API	Application Programming Interface
BMDMA	Block Mode DMA
DAC	Digital-to-Analog Converter
DMA	Direct Memory Access
DMDMA	Demand Mode DMA
GSC	General Standards Corporation
PCI	Peripheral Component Interconnect
PIO	Programmed I/O
PMC	PCI Mezzanine Card
PMC66	This is a PMC formfactor device that can operate at up to 66MHz over the PCI bus.
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 16HSDI4AO4 installation directory or any of its subdirectories.
16HSDI4AO4	This is used as a general reference to any device supported by this driver.
API Library	This is a library that provides application-level access to 16HSDI4AO4 hardware.
Application	This is a user mode process, which runs in user space with user mode privileges.
Driver	This is the 16HSDI4AO4 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 16HSDI4AO4 applications. The overall architecture is illustrated in Figure 1 below.

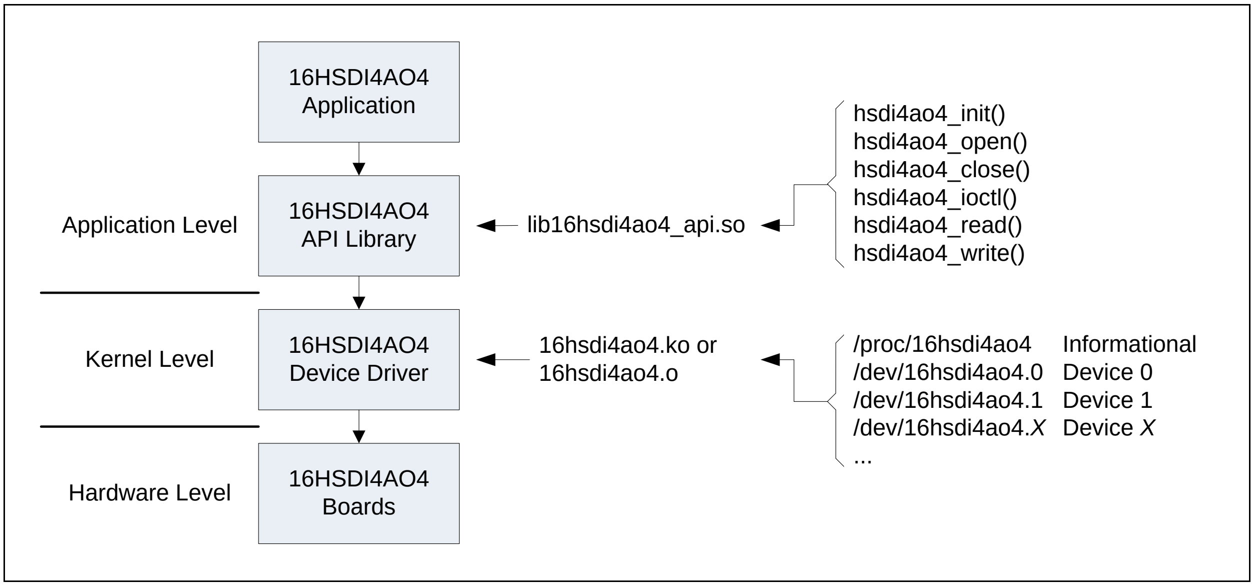

Figure 1 Basic architectural representation.

1.4.2. API Library

The primary means of accessing 16HSDI4AO4 boards is via the 16HSDI4AO4 API Library. This library forms a layer between the application and the driver. Additional information is given in section 4 (page 18). With the library, applications are able to open and close a device and, while open, perform I/O control and read and write operations.

1.4.3. Device Driver

The device driver is the host software that provides a means of communicating directly with 16HSDI4AO4 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 16HSDI4AO4 is a high-performance, 16-bit analog I/O board that incorporates up to four input channels and up to four 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 1M samples per second over each channel. Internal clocking permits sampling rates from 1M samples per second down to near zero samples per second. Onboard storage permits data buffering of up to 256K input samples, for all input channels collectively, and up to 256K output samples, for all output channels collectively, between the cable interface and the PCI bus. This allows the 16HSDI4AO4 to sustain continuous throughput over the cable interface independent of the PCI bus interface. The 16HSDI4AO4 also permits multiple boards to be synchronized so that all boards sample data in unison. In addition, the board includes autocalibration capability, as well as 16 digital I/O signals configurable per byte as input or output.

1.6. Reference Material

The following reference material may be of particular benefit in using the 16HSDI4AO4. The specifications provide the information necessary for an in depth understanding of the specialized features implemented on this board.

- The applicable *16HSDI4AO4 User Manual* from General Standards Corporation.

- The *PCI9056 PCI Bus Master Interface Chip* data handbook from PLX Technology, Inc.

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

1.7. Licensing

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

2. Installation

2.1. CPU and Kernel Support

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

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

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

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

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

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

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

2.1.1. 32-bit Support Under 64-bit Environments

This driver supports 32-bit applications under 64-bit environments. The availability of this feature in the kernel depends on a 64-bit kernel being configured to support 32-bit application compatibility. Additionally, 2.6 kernels prior to 2.6.11 implemented 32-bit compatibility in a way that resulted in some drivers not being able to take advantage of the feature. (In these kernels a driver's IOCTL command codes must be globally unique. Beginning with 2.6.11 this requirement has been lifted.) If the driver is not able to provide 32-bit support under a 64-bit kernel, the "`32-bit support`" field in the `/proc/16hsdi4ao4` file will be "`no`".

2.2. The /proc/ File System

While the driver is running, the text file `/proc/16hsdi4ao4` can be read to obtain information about the driver and the boards it detects. Each file entry includes an entry name followed immediately by a colon, a space character, and the entry value. Below is an example of what appears in the file, followed by descriptions of each entry.

```
version: 2.8.104.47
32-bit support: yes
boards: 1
models: 16HSDI4AO4
```

Entry	Description
`version`	This gives the driver version number in the form `x.x.x.x`.
`32-bit support`	This reports the driver's support for 32-bit applications. This will be either "`yes`" or "`no`" for 64-bit driver builds and "`yes (native)`" for 32-bit builds.
`boards`	This identifies the total number of boards the driver detected.
`models`	This gives a comma separated list of the basic model number for each board the driver detected. The model numbers are listed in the same order that the boards are accessed via the API Library's open function.

2.3. File List

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

File	Description
`16hsdi4ao4.linux.tar.gz`	This archive contains the driver, the API Library and all related files.
`16hsdi4ao4_linux_um.pdf`	This is a PDF version of this user manual, which is included in the archive.

2.4. Directory Structure

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

Directory	Description
`16hsdi4ao4/`	This is the driver root directory. It contains the documentation, the Overall Make Script (section 2.7, page 13) and the below listed subdirectories.
…`/api/`	This directory contains the API Library source files (section 4, page 18).
…`/docsrc/`	This directory contains the source files for the code samples given in this document (section 6, page 58).
…`/driver/`	This directory contains the device driver source files (section 5, page 54).
…`/include/`	This directory contains the header files for the various libraries.
…`/lib/`	This directory contains all of the libraries built from the installed sources.

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

2.5. Installation

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

1. Create and change to the directory where the files are to be installed, such as /usr/src/linux/drivers/. (The path name may vary among distributions and kernel versions.)

2. Copy the archive file 16hsdi4ao4.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 16hsdi4ao4 in the current directory, and then copies all of the archive's files into this new directory.

```
tar –xzvf 16hsdi4ao4.linux.tar.gz
```

2.6. Removal

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

NOTE: The following steps may require elevated privileges.

1. Shutdown the driver as described in section 5.6 (page 57).

2. Change to the directory where the driver archive was installed, which may have been /usr/src/linux/drivers/. (The path name may vary among distributions and kernel versions.)

3. Issue the below command to remove the driver archive and all of the installed driver files.

```
rm -rf 16hsdi4ao4.linux.tar.gz 16hsdi4ao4
```

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

```
rm -f /dev/16hsdi4ao4.*
```

5. If the automatic startup procedure was adopted (section 5.3.2, page 55), then edit the system startup script rc.local and remove the line that invokes the 16HSDI4AO4's start script. The file rc.local should be located in the /etc/rc.d/ directory.

2.7. Overall Make Script

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

NOTE: The following steps may require elevated privileges.

1. Change to the driver root directory (…/16hsdi4ao4/).

2. Remove existing build targets using the below command. This does not unload the driver.

```
./make_all clean
```

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

```
./make_all
```

2.8. Environment Variables

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

2.8.1. GSC_API_COMP_FLAGS

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

Undefined or Empty	`== Compiling: init.c`
	`== Compiling: ioctl.c`
	`== Compiling: open.c`

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

2.8.2. GSC_API_LINK_FLAGS

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

Undefined or Empty	`==== Linking: ../lib/lib16hsdi4ao4_api.so`
Defined and Not Empty	`==== Linking: ../lib/lib16hsdi4ao4_api.so   (added 'xxx')`

2.8.3. GSC_LIB_COMP_FLAGS

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

Undefined or Empty	`== Compiling: close.c`
	`== Compiling: init.c`
	`== Compiling: ioctl.c`

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

2.8.4. GSC_LIB_LINK_FLAGS

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

Undefined or Empty	`==== Linking: ../lib/16hsdi4ao4_utils.a`
Defined and Not Empty	`==== Linking: ../lib/16hsdi4ao4_utils.a   (added 'xxx')`

2.8.5. GSC_APP_COMP_FLAGS

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

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

2.8.6. GSC_APP_LINK_FLAGS

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

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

3. Main Interface Files

This section gives general information on the suggested device interface files to use when developing 16HSDI4AO4 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 16HSDI4AO4 driver installation. For ease of use it is suggested that applications include only the single header file shown below rather than individually including those headers identified separately later in this document. Including this header file pulls in all other pertinent 16HSDI4AO4 specific header files. Therefore, sources may include only this one 16HSDI4AO4 header and make files may reference only this one 16HSDI4AO4 include directory.

Description	File	Location
Header File	`16hsdi4ao4_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 16HSDI4AO4 driver installation. For ease of use it is suggested that applications link only the single library file shown below rather than individually linking those libraries identified separately later in this document. Linking this library file pulls in all other static libraries included with the driver. Therefore, make files may reference only this one 16HSDI4AO4 static library and only this one 16HSDI4AO4 library directory.

Description	File	Location
Static Library	`16hsdi4ao4_main.a`	`…/lib/`
	`16hsdi4ao4_multi.a`	

NOTE: For applications using the 16HSDI4AO4 and no other GSC devices, link the `16hsdi4ao4_main.a` library. For applications using multiple GSC device types, link the `xxxx_main.a` library for one of the devices and the `xxxx_multi.a` library for the others. Linking multiple `xxxx_main.a` libraries may likely produce link errors due to duplicate symbols being defined. While it may make little or no difference, it is recommended that one choose the `xxxx_main.a` library from the driver with the largest number in positions three (x.x.X.x.x) and/or four (x.x.x.X.x) in the driver release version number.

NOTE: The 16HSDI4AO4 API Library is implemented as a shared library and is thus not linked with the 16HSDI4AO4 Main Library. The API Library must be linked with applications by adding the argument `-l16hsdi4ao4_api` to the linker command line.

3.2.1. Build

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

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

2. Remove existing build targets using the below command.

   ```
   make clean
   ```

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

   ```
   make
   ```

3.2.2. System Libraries

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

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

4. API Library

The 16HSDI4AO4 API Library is the software interface between user applications and the 16HSDI4AO4 device driver. The interface is accessed by including the header file 16hsdi4ao4_api.h.

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

4.1. Files

The library files are summarized in the table below.

Description	File	Location
Source Files	*.c, *.h …	…/api/
Header File	16hsdi4ao4_api.h	…/include/
Library File	lib16hsdi4ao4_api.so	…/lib/ /usr/lib/ †

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

4.2. Build

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

NOTE: The following steps may require elevated privileges.

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

2. Remove existing build targets using the below command.

   ```
   make clean
   ```

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

   ```
   make
   ```

4.3. Library Use

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

Description	File	Location	Linker Argument
Header File	16hsdi4ao4_api.h	…/include/	
Shared Object Library	lib16hsdi4ao4_api.so	…/lib/	
		/usr/lib/	-l16hsdi4ao4_api

4.4. Macros

The API Library and driver interfaces include the following macros, which are defined in 16hsdi4ao4.h.

4.4.1. IOCTL Services

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

4.4.2. Registers

The following gives the complete set of 16HSDI4AO4 registers.

4.4.2.1. GSC Registers

The following table gives the complete set of GSC specific 16HSDI4AO4 registers. Please note that the set of registers supported by any given device may vary according to model and firmware version. For the set of supported registers and their detailed definitions refer to the appropriate *16HSDI4AO4 User Manual*.

> **NOTE**: Refer to the output of the "id" sample application (…`/id/`) for a complete list of the registers supported by the device being accessed.

Macro	Description
`HSDI4AO4_GSC_ACR`	Assembly Configuration Register (ACR)
`HSDI4AO4_GSC_AIBR`	Analog Input Buffer Register (AIBR)
`HSDI4AO4_GSC_AOBR`	Analog Output Buffer Register (AOBR)
`HSDI4AO4_GSC_AOC0R`	Analog Output Channel 0 Register (AOC0R)
`HSDI4AO4_GSC_AOC1R`	Analog Output Channel 1 Register (AOC1R)
`HSDI4AO4_GSC_AOC2R`	Analog Output Channel 2 Register (AOC2R)
`HSDI4AO4_GSC_AOC3R`	Analog Output Channel 3 Register (AOC3R)
`HSDI4AO4_GSC_AVR`	Autocal Values Register (AVR)
`HSDI4AO4_GSC_BCR`	Board Control Register (BCR)
`HSDI4AO4_GSC_BOOR`	Buffered Output Operations Register (BOOR)
`HSDI4AO4_GSC_DIOPR`	Digital I/O Port Register (DIOPR)
`HSDI4AO4_GSC_IBSR`	Input Buffer Size Register (IBSR)
`HSDI4AO4_GSC_IBTR`	Input Buffer Threshold Register (IBTR)
`HSDI4AO4_GSC_ICR`	Input Configuration Register (ICR)
`HSDI4AO4_GSC_MCAR`	Master Clock Adjust Register (MCAR)
`HSDI4AO4_GSC_OBSR`	Output Buffer Size Register (OBSR)
`HSDI4AO4_GSC_OBTR`	Output Buffer Threshold Register (OBTR)
`HSDI4AO4_GSC_PSR`	Primary Status Register (PSR)
`HSDI4AO4_GSC_RGAR`	Rate Generator A Register (RGAR)
`HSDI4AO4_GSC_RGBR`	Rate Generator B Register (RGBR)
`HSDI4AO4_GSC_RGCR`	Rate Generator C Register (RGCR)

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 `16hsdi4ao4_api.h`.

4.4.2.3. PLX Feature Set Registers

Access to the PLX registers is seldom required so these registers are not listed here. For the complete list of the PLX register identifiers refer to the driver header file `gsc_pci9056.h`, which is automatically included via `16hsdi4ao4_api.h`.

4.5. Data Types

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

4.6. Functions

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

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

4.6.1. hsdi4ao4_close()

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

Prototype

```
int hsdi4ao4_close(int fd);
```

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

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

Example

```
#include <stdio.h>

#include "16hsdi4ao4_dsl.h"

int hsdi4ao4_close_dsl(int fd)
{
    int errs;
    int ret;

    ret = hsdi4ao4_close(fd);

    if (ret)
        printf("ERROR: hsdi4ao4_close() returned %d\n", ret);

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

4.6.2. hsdi4ao4_init()

This function is the entry point to initializing the 16HSDI4AO4 API Library and must be the first call into the Library. This function may be called more than once, but only the first successful call actually initializes the library. Subsequent calls perform no operation at all. All other API calls return a failure status when the API Library is not initialized.

Prototype

```
int hsdi4ao4_init(void);
```

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

Example

```
#include <stdio.h>

#include "16hsdi4ao4_dsl.h"

int hsdi4ao4_init_dsl(void)
{
    int errs;
    int ret;

    ret = hsdi4ao4_init();

    if (ret)
        printf("ERROR: hsdi4ao4_init() returned %d\n", ret);

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

4.6.3. hsdi4ao4_ioctl()

This function is the entry point to performing setup and control operations on a 16HSDI4AO4. This function should only be called after a successful open of the respective device. The specific operation performed varies according to the `request` argument. The `request` argument also governs the use and interpretation of the `arg` argument. The set of supported options for the `request` argument consists of the IOCTL services supported by the driver, which are defined in section 4.7 (page 25).

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

Prototype

```
int hsdi4ao4_ioctl(int fd, int request, void* arg);
```

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

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

Example

```
#include <stdio.h>

#include "16hsdi4ao4_dsl.h"

int hsdi4ao4_ioctl_dsl(int fd, int request, void* arg)
{
    int errs;
    int ret;

    ret = hsdi4ao4_ioctl(fd, request, arg);

    if (ret)
        printf("ERROR: hsdi4ao4_ioctl() returned %d\n", ret);

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

4.6.4. hsdi4ao4_open()

This function is the entry point to open a connection to a 16HSDI4AO4 board. Before returning, the initialize IOCTL service is called to reset all hardware and software settings to their defaults.

Prototype

```
int hsdi4ao4_open(int device, int share, int* fd);
```

<table>
<tr><th>Argument</th><th>Description</th></tr>
<tr><td>device</td><td>This is the zero-based index of the 16HSDI4AO4 to access. †</td></tr>
<tr><td>share</td><td>Open the device in Shared Access Mode? If non-zero the device is opened in Shared Access Mode (see below). If zero the device is opened in Exclusive Access Mode (see below).</td></tr>
<tr><td>fd</td><td>The device handle is returned here. The pointer cannot be NULL. Values returned are as follows.
<table>
<tr><th>Value</th><th>Description</th></tr>
<tr><td>>= 0</td><td>This is the handle to use to access the device in subsequent calls.</td></tr>
<tr><td>-1</td><td>There was an error. The device is not accessible.</td></tr>
</table></td></tr>
</table>

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

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

Example

```
#include <stdio.h>

#include "16hsdi4ao4_dsl.h"
```

```
int hsdi4ao4_open_dsl(int device, int share, int* fd)
{
    int errs;
    int ret;

    ret = hsdi4ao4_open(device, share, fd);

    if (ret)
        printf("ERROR: hsdi4ao4_open() returned %d\n", ret);

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

4.6.4.1. Access Modes

The value of the `share` argument determines the device access mode, as follows.

Shared Access Mode:

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

Exclusive Access Mode:

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

4.6.5. hsdi4ao4_read()

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

> **NOTE:** If an open was performed using an index of `-1`, then read requests will acquire information from the driver (section 2.2, page 12) rather than data from a device.

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

Prototype

```
int hsdi4ao4_read(int fd, void* dst, size_t bytes);
```

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

Return Value	Description
0 to bytes	The operation succeeded. When reading from a device, a value less than bytes indicates that the I/O timeout period lapsed (section 4.7.64, page 46) before the entire request could be satisfied.
< 0	An error occurred. See error value description above.

Example

```
#include <stdio.h>

#include "16hsdi4ao4_dsl.h"

int hsdi4ao4_read_dsl(int fd, void* dst, size_t bytes, size_t* qty)
{
    int errs;
    int ret;

    ret = hsdi4ao4_read(fd, dst, bytes);

    if (ret < 0)
        printf("ERROR: hsdi4ao4_read() returned %d\n", ret);

    if (qty)
        qty[0]  = (ret < 0) ? 0 : (size_t) ret;

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

4.6.6. hsdi4ao4_write()

This function is the entry point to writing data to an open 16HSDI4AO4. The function writes up to bytes bytes to the device.

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

Prototype

```
int hsdi4ao4_write(int fd, const void* src, size_t bytes);
```

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

Return Value	Description
0 to bytes	The operation succeeded. A value less than bytes indicates that the I/O timeout period lapsed (section 4.7.64, page 46) before the entire request could be satisfied.
< 0	An error occurred. See error value description above.

Example

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

```
#include "16hsdi4ao4_dsl.h"

int  hsdi4ao4_write_dsl(int  fd,  const  void*  src,  size_t  bytes,
size_t* qty)
{
    int errs;
    int ret;

    ret = hsdi4ao4_write(fd, src, bytes);

    if (ret < 0)
        printf("ERROR: hsdi4ao4_write() returned %d\n", ret);

    if (qty)
        qty[0]  = (ret < 0) ? 0 : (size_t) ret;

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

4.7. IOCTL Services

The 16HSDI4AO4 API Library and device driver implement the following IOCTL services. Each service is described along with the applicable `hsdi4ao4_ioctl()` function arguments.

4.7.1. HSDI4AO4_IOCTL_AI_BUF_CLEAR

This service immediately clears the current content from the input buffer. It also clears the associated overflow and underflow status bits. This service does not halt sampling.

Usage

Argument	Description
request	HSDI4AO4_IOCTL_AI_BUF_CLEAR
arg	Not used.

4.7.2. HSDI4AO4_IOCTL_AI_BUF_LEVEL

This service returns the current number of 32-bit data items in the input buffer.

Usage

Argument	Description
request	HSDI4AO4_IOCTL_AI_BUF_LEVEL
arg	s32*

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

4.7.3. HSDI4AO4_IOCTL_AI_BUF_OVERFLOW

This service operates on the Input Buffer Overflow status.

NOTE: This service always returns the current overflow status.

Usage

Argument	Description
request	HSDI4AO4_IOCTL_AI_BUF_OVERFLOW
arg	s32*

Valid argument values supplied to the service are as follows.

Value	Description
-1	Retrieve the current state.
HSDI4AO4_BUF_OVERFLOW_CLEAR	Clear the overflow status.
HSDI4AO4_BUF_OVERFLOW_TEST	Report the current status.

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

Value	Description
HSDI4AO4_BUF_OVERFLOW_NO	The buffer has not experienced an overflow condition.
HSDI4AO4_BUF_OVERFLOW_YES	The buffer has experienced an overflow condition.

4.7.4. HSDI4AO4_IOCTL_AI_BUF_THR_LVL

This service configures the input buffer threshold level.

Usage

Argument	Description
request	HSDI4AO4_IOCTL_AI_BUF_THR_LVL
arg	s32*

Valid argument values are from zero to 0x7FFFF, and -1. A value of -1 will return the current threshold level setting.

4.7.5. HSDI4AO4_IOCTL_AI_BUF_THR_STS

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

Usage

Argument	Description
request	HSDI4AO4_IOCTL_AI_BUF_THR_STS
arg	s32*

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

Value	Description
HSDI4AO4_BUF_THR_STS_CLEAR	The input buffer contains Threshold Level number of data items, or fewer.
HSDI4AO4_BUF_THR_STS_SET	The input buffer contains more than Threshold Level number of data items.

4.7.6. HSDI4AO4_IOCTL_AI_BUF_UNDERFLOW

This service operates on the Input Buffer Underflow status.

NOTE: This service always returns the current underflow status.

Usage

Argument	Description
request	HSDI4AO4_IOCTL_AI_BUF_UNDERFLOW
arg	s32*

Valid argument values supplied to the service are as follows.

Value	Description
-1	Retrieve the current state.
HSDI4AO4_BUF_UNDERFLOW_CLEAR	Clear the underflow status.
HSDI4AO4_BUF_UNDERFLOW_TEST	Report the current status.

Valid argument values are as follows.

Value	Description
HSDI4AO4_BUF_UNDERFLOW_NO	The buffer has experienced an underflow condition.
HSDI4AO4_BUF_UNDERFLOW_YES	The buffer has not experienced an underflow condition.

4.7.7. HSDI4AO4_IOCTL_AI_BURST_ENABLE

This service enables and disables input bursting.

Usage

Argument	Description
request	HSDI4AO4_IOCTL_AI_BURST_ENABLE
arg	s32*

Valid argument values are as follows.

Value	Description
-1	Retrieve the current setting.
HSDI4AO4_BURST_ENABLE_NO	This option disables input bursting.
HSDI4AO4_BURST_ENABLE_YES	This option enables input bursting.

4.7.8. HSDI4AO4_IOCTL_AI_BURST_SIZE

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

Usage

Argument	Description
request	HSDI4AO4_IOCTL_AI_BURST_SIZE
arg	s32*

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

4.7.9. HSDI4AO4_IOCTL_AI_BURST_STATUS

This service reports on the board’s input burst status.

Usage

Argument	Description
request	HSDI4AO4_IOCTL_AI_BURST_STATUS
arg	s32*

The value returned will be one of the following.

Value	Description
HSDI4AO4_BURST_STATUS_BUSY	The board is not ready to start an input burst operation.
HSDI4AO4_BURST_STATUS_READY	The board is ready to start an input burst operation.

4.7.10. HSDI4AO4_IOCTL_AI_CHAN_SEL

This service configures the set of active 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	HSDI4AO4_IOCTL_AI_CHAN_SEL
arg	s32*

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

4.7.11. HSDI4AO4_IOCTL_AI_CLOCK_MODE

This service configures the analog input clock operating mode.

Usage

Argument	Description
request	HSDI4AO4_IOCTL_AI_CLOCK_MODE
arg	s32*

Valid argument values are as follows.

Value	Description
-1	Retrieve the current setting.
HSDI4AO4_CLOCK_MODE_IN	The input clock signal is an input.
HSDI4AO4_CLOCK_MODE_OUT	The input clock signal is an output.

4.7.12. HSDI4AO4_IOCTL_AI_ENABLE

This service enables or disables Analog Input sampling.

Usage

Argument	Description
request	HSDI4AO4_IOCTL_AI_ENABLE
arg	s32*

Valid argument values are as follows.

Value	Description
-1	Retrieve the current setting.
HSDI4AO4_AIO_ENABLE_NO	This disables input sampling.
HSDI4AO4_AIO_ENABLE_YES	This enables input samplings.

4.7.13. HSDI4AO4_IOCTL_AI_MODE

This service configures the board's Analog Input Mode.

Usage

Argument	Description
request	HSDI4AO4_IOCTL_AI_MODE
arg	s32*

Valid argument values are as follows.

Value	Description
-1	Retrieve the current setting.
HSDI4AO4_AIN_MODE_AO_0	Route output channel zero to the inputs.
HSDI4AO4_AIN_MODE_AO_1	Route output channel one to the inputs.
HSDI4AO4_AIN_MODE_AO_2	Route output channel two to the inputs.
HSDI4AO4_AIN_MODE_AO_3	Route output channel three to the inputs.
HSDI4AO4_AIN_MODE_DIFF	Configure the input channels for differential operation.
HSDI4AO4_AIN_MODE_SINGLE	Configure the input channels for single-ended operation.
HSDI4AO4_AIN_MODE_VREF	Configure the input channels for +VREF input testing
HSDI4AO4_AIN_MODE_ZERO	Configure the input channels for Zero input testing

4.7.14. HSDI4AO4_IOCTL_AI_RANGE

This service configures the analog input voltage range.

Usage

Argument	Description
request	HSDI4AO4_IOCTL_AI_RANGE
arg	s32*

Valid argument values are as follows.

Value	Description
-1	Retrieve the current setting.
HSDI4AO4_RANGE_2_5V	Set the input voltage range to ±2.5 volts.
HSDI4AO4_RANGE_5V	Set the input voltage range to ±5 volts.
HSDI4AO4_RANGE_10V	Set the input voltage range to ±10 volts.

4.7.15. HSDI4AO4_IOCTL_AI_SW_CLOCK

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

Usage

Argument	Description
request	HSDI4AO4_IOCTL_AI_SW_CLOCK
arg	Not used.

4.7.16. HSDI4AO4_IOCTL_AI_SW_TRIGGER

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

Usage

Argument	Description
request	HSDI4AO4_IOCTL_AI_SW_TRIGGER
arg	Not used.

4.7.17. HSDI4AO4_IOCTL_AO_BUF_CLEAR

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

Usage

Argument	Description
request	HSDI4AO4_IOCTL_AO_BUF_CLEAR
arg	Not used.

4.7.18. HSDI4AO4_IOCTL_AO_BUF_EMPTY

This service reports on the output buffer’s empty state.

Usage

Argument	Description
request	HSDI4AO4_IOCTL_AO_BUF_EMPTY
arg	s32*

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

Value	Description
HSDI4AO4_AO_BUFFER_EMPTY	The output buffer is empty.
HSDI4AO4_AO_BUFFER_NOT_EMPTY	The output buffer is not empty.

4.7.19. HSDI4AO4_IOCTL_AO_BUF_LEVEL

This service returns the current number of 32-bit data items in the output buffer.

Usage

Argument	Description
request	HSDI4AO4_IOCTL_AO_BUF_LEVEL
arg	s32*

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

4.7.20. HSDI4AO4_IOCTL_AO_BUF_LOAD_REQ

This service requests load access to the output buffer. The driver waits for access to be granted. The waiting period limit is the current Tx I/O timeout period.

Usage

Argument	Description
request	HSDI4AO4_IOCTL_AO_BUF_LOAD_REQ
arg	Not used.

4.7.21. HSDI4AO4_IOCTL_AO_BUF_LOAD_STS

This service retrieves the output buffer load access status.

Usage

Argument	Description
request	HSDI4AO4_IOCTL_AO_BUF_LOAD_STS
arg	s32*

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

Value	Description
HSDI4AO4_BUF_LOAD_STS_NOT_READY	The output buffer is not yet accessible.
HSDI4AO4_BUF_LOAD_STS_READY	The output buffer is available for output data.

4.7.22. HSDI4AO4_IOCTL_AO_BUF_MODE

This service operates on the output buffer operating mode.

Usage

Argument	Description
request	HSDI4AO4_IOCTL_AO_BUF_MODE
arg	s32*

Valid argument values are as follows.

Value	Description
-1	Retrieve the current setting.
HSDI4AO4_BUF_MODE_CIRC	After data exits the buffer, it is routed back into the buffer.
HSDI4AO4_BUF_MODE_OPEN	Data is not routed back into the output buffer.

4.7.23. HSDI4AO4_IOCTL_AO_BUF_OVER_DATA

This service operates on the output buffer data overflow status.

NOTE: This service always returns the current data overflow status.

Usage

Argument	Description
request	HSDI4AO4_IOCTL_AO_BUF_OVER_DATA
arg	s32*

Valid argument values supplied to the service are as follows.

Value	Description
-1	Retrieve the current state.
HSDI4AO4_BUF_OVERFLOW_CLEAR	Clear the overflow status.
HSDI4AO4_BUF_OVERFLOW_TEST	Report the current status.

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

Value	Description
HSDI4AO4_BUF_OVERFLOW_NO	The buffer has not experienced an overflow condition.
HSDI4AO4_BUF_OVERFLOW_YES	The buffer has experienced an overflow condition.

4.7.24. HSDI4AO4_IOCTL_AO_BUF_OVER_FRAM

This service operates on the output buffer frame overflow status.

NOTE: This service always returns the current frame overflow status.

Usage

Argument	Description
request	HSDI4AO4_IOCTL_AO_BUF_OVER_FRAM
arg	s32*

Valid argument values supplied to the service are as follows.

Value	Description
-1	Retrieve the current state.
HSDI4AO4_BUF_OVERFLOW_CLEAR	Clear the overflow status.
HSDI4AO4_BUF_OVERFLOW_TEST	Report the current status.

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

Value	Description
HSDI4AO4_BUF_OVERFLOW_NO	The buffer has not experienced an overflow condition.
HSDI4AO4_BUF_OVERFLOW_YES	The buffer has experienced an overflow condition.

4.7.25. HSDI4AO4_IOCTL_AO_BUF_THR_LVL

This service configures the output buffer threshold level.

Usage

Argument	Description
request	HSDI4AO4_IOCTL_AO_BUF_THR_LVL
arg	s32*

Valid argument values are from zero to 0x7FFFF, and -1. A value of -1 will return the current threshold level setting.

4.7.26. HSDI4AO4_IOCTL_AO_BUF_THR_STS

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

Usage

Argument	Description
request	HSDI4AO4_IOCTL_AO_BUF_THR_STS
arg	s32*

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

Value	Description
HSDI4AO4_BUF_THR_STS_CLEAR	The output buffer contains Threshold Level number of data items, or fewer.
HSDI4AO4_BUF_THR_STS_SET	The output buffer contains more than Threshold Level number of data items.

4.7.27. HSDI4AO4_IOCTL_AO_BURST_ENABLE

This service enables or disables output bursting.

Usage

Argument	Description
request	HSDI4AO4_IOCTL_AO_BURST_ENABLE
arg	s32*

Valid argument values are as follows.

Value	Description
-1	Retrieve the current setting.
HSDI4AO4_BURST_ENABLE_NO	This option disables input bursting.
HSDI4AO4_BURST_ENABLE_YES	This option enables input bursting.

4.7.28. HSDI4AO4_IOCTL_AO_BURST_STATUS

This service reports on the board's output burst readiness status.

Usage

Argument	Description
request	HSDI4AO4_IOCTL_AO_BURST_STATUS
arg	s32*

The value returned will be one of the following.

Value	Description
HSDI4AO4_BURST_STATUS_BUSY	The board is not ready to start an output burst operation.
HSDI4AO4_BURST_STATUS_READY	The board is ready to start an output burst operation.

4.7.29. HSDI4AO4_IOCTL_AO_CHAN_SEL

This service configures the set of active 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	HSDI4AO4_IOCTL_AO_CHAN_SEL
arg	s32*

Valid argument values are from zero to `0xF`, for four channel boards only, or `-1` to retrieve the current setting.

4.7.30. HSDI4AO4_IOCTL_AO_CLOCK_MODE

This service configures the source for the output sample clock.

Usage

Argument	Description
request	HSDI4AO4_IOCTL_AO_CLOCK_MODE
arg	s32*

Valid argument values are as follows.

Value	Description
-1	Retrieve the current setting.
HSDI4AO4_AO_CLOCK_MODE_IN	The analog output clock cable signal is an input.
HSDI4AO4_AO_CLOCK_MODE_OUT	The analog output clock cable signal is an output.

4.7.31. HSDI4AO4_IOCTL_AO_CLOCK_STATUS

This service reports on the board's output clock status.

Usage

Argument	Description
request	HSDI4AO4_IOCTL_AO_CLOCK_STATUS
arg	s32*

The value returned will be one of the following.

Value	Description
HSDI4AO4_CLOCK_STATUS_NOT_READY	Output clocking is not ready.
HSDI4AO4_CLOCK_STATUS_READY	Output clocking is ready.

4.7.32. HSDI4AO4_IOCTL_AO_ENABLE

This service enables or disables output sampling.

Usage

Argument	Description
request	HSDI4AO4_IOCTL_AO_ENABLE
arg	s32*

Valid argument values are as follows.

Value	Description
-1	Retrieve the current setting.
HSDI4AO4_AIO_ENABLE_NO	This disables output sampling.
HSDI4AO4_AIO_ENABLE_YES	This enables output samplings.

4.7.33. HSDI4AO4_IOCTL_AO_MODE

This service configures the board's analog output mode.

Usage

Argument	Description
request	HSDI4AO4_IOCTL_AO_MODE
arg	s32*

Valid argument values are as follows.

Value	Description
-1	Retrieve the current setting.
HSDI4AO4_AO_MODE_FIFO	Data is processed through the output buffer via the `hsdi4ao4_write()` call.
HSDI4AO4_AO_MODE_REG	Data is processed through the output registers via the IOCTL based register access services.

4.7.34. HSDI4AO4_IOCTL_AO_OUTPUT_MODE

This service configures the timing of how output data appears at the cable interface.

Usage

Argument	Description
request	HSDI4AO4_IOCTL_AO_OUTPUT_MODE
arg	s32*

Valid argument values are as follows.

Value	Description
-1	Retrieve the current setting.
HSDI4AO4_AO_OUTPUT_MODE_IMMED	The output appears immediately.
HSDI4AO4_AO_OUTPUT_MODE_SIMUL	Output for all active channels appears simultaneously.

4.7.35. HSDI4AO4_IOCTL_AO_RANGE

This service configures the analog output voltage range.

Usage

Argument	Description
request	HSDI4AO4_IOCTL_AO_RANGE
arg	s32*

Valid argument values are as follows.

Value	Description
-1	Retrieve the current setting.
HSDI4AO4_RANGE_2_5V	Set the output voltage range to ±2.5 volts.
HSDI4AO4_RANGE_5V	Set the output voltage range to ±5 volts.
HSDI4AO4_RANGE_10V	Set the output voltage range to ±10 volts.

4.7.36. HSDI4AO4_IOCTL_AO_SW_CLOCK

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

Usage

Argument	Description
request	HSDI4AO4_IOCTL_AO_SW_CLOCK
arg	Not used.

4.7.37. HSDI4AO4_IOCTL_AO_SW_TRIGGER

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

Usage

Argument	Description
request	HSDI4AO4_IOCTL_AO_SW_TRIGGER
arg	Not used.

4.7.38. HSDI4AO4_IOCTL_AO_SYNC_MODE

This service configures the signal direction of the cable's trigger signal.

Usage

Argument	Description
request	HSDI4AO4_IOCTL_AO_SYNC_MODE
arg	s32*

Valid argument values are as follows.

Value	Description
-1	Retrieve the current setting.
HSDI4AO4_AO_SYNC_MODE_IND	The output is independent of the input.
HSDI4AO4_AO_SYNC_MODE_AI_SAMP	The output is synchronized to the input sampling clock.
HSDI4AO4_AO_SYNC_MODE_AI_ACQ	The output is synchronized to the acquisition of input data.

4.7.39. HSDI4AO4_IOCTL_AUTOCAL

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

> **NOTE:** This service overwrites the current interrupt selection in order to detect the Autocalibration Done interrupt.
>
> **NOTE:** When an error is encountered, the service writes a brief, descriptive error message to the system log.

Usage

Argument	Description
request	HSDI4AO4_IOCTL_AUTOCAL
arg	Not used.

4.7.40. HSDI4AO4_IOCTL_AUTOCAL_STATUS

This service reports the status of the most recent autocalibration cycle.

Usage

Argument	Description
request	HSDI4AO4_IOCTL_AUTOCAL_STATUS
arg	s32*

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

Value	Description
HSDI4AO4_AUTOCAL_STATS_ACTIVE	Autocalibration is in progress.
HSDI4AO4_AUTOCAL_STATS_FAIL	Autocalibration failed.
HSDI4AO4_AUTOCAL_STATS_PASS	Autocalibration passed.

4.7.41. HSDI4AO4_IOCTL_DATA_FORMAT

This service sets the data encoding format.

Usage

Argument	Description
request	HSDI4AO4_IOCTL_DATA_FORMAT
arg	s32*

Valid argument values are as follows.

Value	Description
-1	Retrieve the current setting.
HSDI4AO4_DATA_FORMAT_2S_COMP	Select the Twos Compliment data format.
HSDI4AO4_DATA_FORMAT_OFF_BIN	Select the Offset Binary encoding format.

4.7.42. HSDI4AO4_IOCTL_DIO_DIR_OUT

This service configures the direction of the digital I/O cable signals. If a bit is set in the bitmap value, then that digital I/O byte’s signals are outputs. The signals are otherwise inputs. Bit position zero corresponds to the byte zero signals.

Usage

Argument	Description
request	HSDI4AO4_IOCTL_DIO_DIR_OUT
arg	s32*

Valid argument values are as follows.

Value	Description
-1	Retrieve the current setting.
HSDI4AO4_DIO_DIR_OUT_B0_IN_B1_IN	Both bytes are inputs.
HSDI4AO4_DIO_DIR_OUT_B0_OT_B1_IN	Byte zero is an output and byte one is an input.
HSDI4AO4_DIO_DIR_OUT_B0_IN_B1_OT	Byte zero is an input and byte one is an output.
HSDI4AO4_DIO_DIR_OUT_B0_OT_B1_OT	Both bytes are outputs.

4.7.43. HSDI4AO4_IOCTL_DIO_READ

This service retrieves the signal levels for the 16 digital I/O lines. Bit position zero corresponds to signal zero.

Usage

Argument	Description
request	HSDI4AO4_IOCTL_DIO_READ
arg	s32*

Argument values returned are from zero to 0xFFFF.

4.7.44. HSDI4AO4_IOCTL_DIO_WRITE

This service applies values to the digital I/O cable signals. The value written is retained even for those signals configured as inputs. Bit position zero corresponds to signal zero.

Usage

Argument	Description
request	HSDI4AO4_IOCTL_DIO_WRITE
arg	s32*

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

4.7.45. HSDI4AO4_IOCTL_GEN_A_ENABLE

This service enables or disables Rate Generator A, which is used for input sampling.

Usage

Argument	Description
request	HSDI4AO4_IOCTL_GEN_A_ENABLE
arg	s32*

Valid argument values are as follows.

Value	Description
-1	Retrieve the current setting.
HSDI4AO4_GEN_ENABLE_NO	This option disables the rate generator.
HSDI4AO4_GEN_ENABLE_YES	This option enables the rate generator.

4.7.46. HSDI4AO4_IOCTL_GEN_A_NDIV

This service sets the Rate Generator A NDIV divider value.

Usage

Argument	Description
request	HSDI4AO4_IOCTL_GEN_A_NDIV
arg	s32*

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

4.7.47. HSDI4AO4_IOCTL_GEN_A_NREF

This service sets the Rate Generator A NREF PLL control value.

Usage

Argument	Description
request	HSDI4AO4_IOCTL_GEN_A_NREF
arg	s32*

Valid argument values are in the range from 30 to 1000, and -1. The value -1 is used to retrieve the current setting.

4.7.48. HSDI4AO4_IOCTL_GEN_A_NVCO

This service sets the Rate Generator A NVCO PLL control value.

Usage

Argument	Description
request	HSDI4AO4_IOCTL_GEN_A_NVCO
arg	s32*

Valid argument values are in the range from 30 to 1000, and -1. The value -1 is used to retrieve the current setting.

4.7.49. HSDI4AO4_IOCTL_GEN_B_ENABLE

This service enables or disables Rate Generator B, which is used for bursting.

Usage

Argument	Description
request	HSDI4AO4_IOCTL_GEN_B_ENABLE
arg	s32*

Valid argument values are as follows.

Value	Description
-1	Retrieve the current setting.
HSDI4AO4_GEN_ENABLE_NO	This option disables the rate generator.
HSDI4AO4_GEN_ENABLE_YES	This option enables the rate generator.

4.7.50. HSDI4AO4_IOCTL_GEN_B_NDIV

This service sets the Rate Generator B NDIV divider value.

Usage

Argument	Description
request	HSDI4AO4_IOCTL_GEN_B_NDIV
arg	s32*

Valid argument values are in the range from 40 to 0xFFFFFF, and -1. The value -1 is used to retrieve the current setting.

4.7.51. HSDI4AO4_IOCTL_GEN_C_ENABLE

This service enables or disables Rate Generator C, which is used for output sampling.

Usage

Argument	Description
request	HSDI4AO4_IOCTL_GEN_C_ENABLE
arg	s32*

Valid argument values are as follows.

Value	Description
-1	Retrieve the current setting.
HSDI4AO4_GEN_ENABLE_NO	This option disables the rate generator.
HSDI4AO4_GEN_ENABLE_YES	This option enables the rate generator.

4.7.52. HSDI4AO4_IOCTL_GEN_C_NDIV

This service sets the Rate Generator C NDIV divider value.

Usage

Argument	Description
request	HSDI4AO4_IOCTL_GEN_C_NDIV
arg	s32*

Valid argument values are in the range from 40 to 0xFFFFFF, and -1. The value -1 is used to retrieve the current setting.

4.7.53. HSDI4AO4_IOCTL_INITIALIZE

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

NOTE: When an error is encountered, the service writes a brief, descriptive error message to the system log.

Usage

Argument	Description
request	HSDI4AO4_IOCTL_INITIALIZE
arg	Not used.

4.7.54. HSDI4AO4_IOCTL_IRQ_ENABLE

This service enables a specified set of interrupts. The interrupts remain enabled until disabled by the application.

Usage

Argument	Description
request	HSDI4AO4_IOCTL_IRQ_ENABLE
arg	s32*

Valid argument values are as follows.

Value	Description
-1	Retrieve the current setting.
HSDI4AO4_IRQ_AI_CLOCK	This refers to the analog input clock.
HSDI4AO4_IRQ_AI_BURST_DONE	This refers to the completion of an analog input burst.
HSDI4AO4_IRQ_AI_BURST_START	This refers to the beginning of an analog input burst.
HSDI4AO4_IRQ_AI_FAULT	This refers to an input buffer overflow or underflow.
HSDI4AO4_IRQ_AI_THRESH_H2L	This refers to a low to high transition of the analog input threshold level status.
HSDI4AO4_IRQ_AI_THRESH_L2H	This refers to a high to low transition of the analog input threshold level status.

HSDI4AO4_IRQ_AO_BURST_READY	This refers to readiness of the board to begin an analog output burst.
HSDI4AO4_IRQ_AO_CLOCK	This refers to the analog output clock.
HSDI4AO4_IRQ_AO_FAULT	This refers to an analog output buffer underflow, or to a data or frame overflow.
HSDI4AO4_IRQ_AO_LOAD_RDY_H2L	This refers to a high to low transition of the output buffer's Load Ready status.
HSDI4AO4_IRQ_AO_LOAD_RDY_L2H	This refers to a low to high transition of the output buffer's Load Ready status.
HSDI4AO4_IRQ_AO_THRESH_H2L	This refers to a high to low transition of the output buffer's threshold level status.
HSDI4AO4_IRQ_AO_THRESH_L2H	This refers to a low to high transition of the output buffer's threshold level status.
HSDI4AO4_IRQ_AUTOCAL_DONE	This refers to the completion of an autocalibration cycle.
HSDI4AO4_IRQ_DIO_0_L2H	This refers to a low to high transition at digital I/O signal D0.

4.7.55. HSDI4AO4_IOCTL_MASTER_CLOCK

This service sets the Master Clock frequency for those occasions in which the driver is not able to correctly identify the board's master clock frequency. The value is expressed in hertz.

Usage

Argument	Description
request	HSDI4AO4_IOCTL_MASTER_CLOCK
arg	s32*

Valid argument values are in the range from `30,000,000` to `47,000,000`, and `-1`. A value of `-1` is used to retrieve the current setting.

4.7.56. HSDI4AO4_IOCTL_MASTER_CLOCK_ADJ

This service configures the Master Clock adjustment setting.

Usage

Argument	Description
request	HSDI4AO4_IOCTL_MASTER_CLOCK_ADJ
arg	s32*

Valid argument values are in the range from `0x0000` to `0xFFFF`, and `-1`. A value of `-1` is used to retrieve the current setting.

4.7.57. HSDI4AO4_IOCTL_QUERY

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

Usage

Argument	Description
request	HSDI4AO4_IOCTL_QUERY
arg	s32*

Valid argument values are as follows.

Value	Description
HSDI4AO4_QUERY_AUTOCAL_MS	This returns the maximum duration of the Autocalibration cycle in milliseconds.
HSDI4AO4_QUERY_CHANNEL_AI_MAX	This returns the maximum number of input channels supported by the board, which may be more that the board's current configuration.
HSDI4AO4_QUERY_CHANNEL_AI_QTY	This returns the actual number of input channels on the current board. If the value returned is -1, then the driver was unable to determine the number of channels.
HSDI4AO4_QUERY_CHANNEL_AO_MAX	This returns the maximum number of output channels supported by the board, which may be more that the board's current configuration.
HSDI4AO4_QUERY_CHANNEL_AO_QTY	This returns the actual number of output channels on the current board. If the value returned is -1, then the driver was unable to determine the number of channels.
HSDI4AO4_QUERY_COUNT	This returns the number of query options supported by the IOCTL service.
HSDI4AO4_QUERY_DEVICE_TYPE	This returns the identifier value for the board's type. This should be GSC_DEV_TYPE_16HSDI4AO4.
HSDI4AO4_QUERY_FIFO_SIZE_RX	This returns the size of the input buffer in 32-bit A/D values.
HSDI4AO4_QUERY_FIFO_SIZE_TX	This returns the size of the output buffer in 32-bit D/A values.
HSDI4AO4_QUERY_FSAMP_MAX	This gives the maximum FSAMP value in S/S for both input and output channels.
HSDI4AO4_QUERY_FSAMP_MIN	This gives the minimum FSAMP value in S/S for both input and output channels.
HSDI4AO4_QUERY_GEN_A_MAX	This is the maximum Rate Generate A output frequency.
HSDI4AO4_QUERY_GEN_A_MAX	This is the minimum Rate Generate A output frequency.
HSDI4AO4_QUERY_GEN_A_NDIV_MAX	This is the maximum Rate Generate A NDIV value.
HSDI4AO4_QUERY_GEN_A_NDIV_MIN	This is the minimum Rate Generate A NDIV value.
HSDI4AO4_QUERY_GEN_A_NREF_MAX	This is the maximum Rate Generate A NREF value.
HSDI4AO4_QUERY_GEN_A_NREF_MIN	This is the minimum Rate Generate A NREF value.
HSDI4AO4_QUERY_GEN_A_NVCO_MAX	This is the maximum Rate Generate A NVCO value.
HSDI4AO4_QUERY_GEN_A_NVCO_MIN	This is the minimum Rate Generate A NVCO value.
HSDI4AO4_QUERY_GEN_B_NDIV_MAX	This is the maximum Rate Generate B NDIV value.
HSDI4AO4_QUERY_GEN_B_NDIV_MIN	This is the minimum Rate Generate B NDIV value.
HSDI4AO4_QUERY_GEN_C_NDIV_MAX	This is the maximum Rate Generate C NDIV value.
HSDI4AO4_QUERY_GEN_C_NDIV_MIN	This is the minimum Rate Generate C NDIV value.
HSDI4AO4_QUERY_INIT_MS	This returns the duration of a board initialization in milliseconds.
HSDI4AO4_QUERY_MASTER_CLOCK_DEF	This returns the default master clock frequency in hertz.
HSDI4AO4_QUERY_MCAR	This indicates if the Master Clock Adjust Register (MCAR) is available.

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

Value	Description
HSDI4AO4_IOCTL_QUERY_ERROR	Either there was a processing error or the query option is unrecognized.

4.7.58. HSDI4AO4_IOCTL_REG_MOD

This service performs a read-modify-write of a 16HSDI4AO4 register. This includes only the GSC firmware registers. The PCI and PLX Feature Set Registers are read-only. Refer to `16hsdi4ao4.h` for the complete list of GSC firmware registers.

Usage

Argument	Description
request	HSDI4AO4_IOCTL_REG_MOD
arg	gsc_reg_t*

Definition

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

Fields	Description
reg	This is set to the identifier for the register to access.
value	This contains the value for the register bits to modify.
mask	This specifies the set of bits to modify. If a bit here is set, then the respective register bits is modified. If a bit here is zero, then the respective register bit is unmodified.

4.7.59. HSDI4AO4_IOCTL_REG_READ

This service reads the value of a 16HSDI4AO4 register. This includes the PCI registers, the PLX Feature Set Registers and the GSC firmware registers. Refer to `16hsdi4ao4.h` and `gsc_pci9056.h` for the complete list of accessible registers.

Usage

Argument	Description
request	HSDI4AO4_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.

4.7.60. HSDI4AO4_IOCTL_REG_WRITE

This service writes a value to a 16HSDI4AO4 register. This includes only the GSC firmware registers. The PCI and PLX Feature Set Registers are read-only. Refer to `16hsdi4ao4.h` for a complete list of the GSC firmware registers.

Usage

Argument	Description
`request`	`HSDI4AO4_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.

4.7.61. HSDI4AO4_IOCTL_RX_IO_ABORT

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

Usage

Argument	Description
`request`	`HSDI4AO4_IOCTL_RX_IO_ABORT`
`arg`	`s32*`

The results are reported as one of the following values.

Value	Description
`HSDI4AO4_IO_ABORT_NO`	A `read()` request was not aborted as none were ongoing.
`HSDI4AO4_IO_ABORT_YES`	An ongoing `read()` request was aborted.

4.7.62. HSDI4AO4_IOCTL_RX_IO_MODE

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

Usage

Argument	Description
request	HSDI4AO4_IOCTL_RX_IO_MODE
arg	s32*

Valid argument values are as follows.

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

4.7.63. HSDI4AO4_IOCTL_RX_IO_OVERFLOW

This service configures the read service to check for an 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	HSDI4AO4_IOCTL_RX_IO_OVERFLOW
arg	s32*

Valid argument values are as follows.

Value	Description
-1	Retrieve the current setting.
HSDI4AO4_IO_OVERFLOW_CHECK	Perform the check. This is the default.
HSDI4AO4_IO_OVERFLOW_IGNORE	Do not perform the check.

4.7.64. HSDI4AO4_IOCTL_RX_IO_TIMEOUT

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

Usage

Argument	Description
request	HSDI4AO4_IOCTL_RX_IO_TIMEOUT
arg	s32*

Valid argument values are in the range from zero to 3600, -1, and HSDI4AO4_IO_TIMEOUT_INFINITE. A value of zero tells the driver not to sleep in order to wait for more data, and should only be used with PIO mode reads. A value of -1 is used to retrieve the current setting. If the option HSDI4AO4_IO_TIMEOUT_INFINITE is used, then the driver will wait indefinitely rather than timing out. The default is 10 seconds.

4.7.65. HSDI4AO4_IOCTL_RX_IO_UNDERFLOW

This service configures the read service to check for an input buffer underflow before performing the read operation. 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`	`HSDI4AO4_IOCTL_RX_IO_UNDERFLOW`
`arg`	`s32*`

Valid argument values are as follows.

Value	Description
`-1`	Retrieve the current setting.
`HSDI4AO4_IO_UNDERFLOW_CHECK`	Perform the check. This is the default.
`HSDI4AO4_IO_UNDERFLOW_IGNORE`	Do not perform the check.

4.7.66. HSDI4AO4_IOCTL_TRIGGER_MODE

This service configures the signal direction of the cable’s trigger signal.

Usage

Argument	Description
`request`	`HSDI4AO4_IOCTL_TRIGGER_MODE`
`arg`	`s32*`

Valid argument values are as follows.

Value	Description
`-1`	Retrieve the current setting.
`HSDI4AO4_TRIGGER_MODE_IN`	The trigger cable signal is an input.
`HSDI4AO4_TRIGGER_MODE_OUT`	The trigger cable signal is an output.

4.7.67. HSDI4AO4_IOCTL_TX_IO_ABORT

This service aborts an ongoing `hsdi4ao4_write()` request.

Usage

Argument	Description
`request`	`HSDI4AO4_IOCTL_TX_IO_ABORT`
`arg`	`s32*`

The results are reported as one of the following values.

Value	Description
HSDI4AO4_IO_ABORT_NO	An hsdi4ao4_write() request was not aborted as none were ongoing.
HSDI4AO4_IO_ABORT_YES	An ongoing hsdi4ao4_write() request was aborted.

4.7.68. HSDI4AO4_IOCTL_TX_IO_MODE

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

Usage

Argument	Description
request	HSDI4AO4_IOCTL_TX_IO_MODE
arg	s32*

Valid argument values are as follows.

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

4.7.69. HSDI4AO4_IOCTL_TX_IO_OVERFLOW

This service configures the write service to check for an output buffer overflow and an output frame overflow 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	HSDI4AO4_IOCTL_TX_IO_OVERFLOW
arg	s32*

Valid argument values are as follows.

Value	Description
-1	Retrieve the current setting.
HSDI4AO4_IO_OVERFLOW_CHECK	Perform the check. This is the default.
HSDI4AO4_IO_OVERFLOW_IGNORE	Do not perform the check.

4.7.70. HSDI4AO4_IOCTL_TX_IO_TIMEOUT

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

Usage

Argument	Description
request	HSDI4AO4_IOCTL_TX_IO_TIMEOUT
arg	s32*

Valid argument values are in the range from zero to 3600, -1, and HSDI4AO4_IO_TIMEOUT_INFINITE. A value of zero tells the driver not to sleep in order to wait for more space, and should only be used with PIO mode reads. A value of -1 is used to retrieve the current setting. If the option HSDI4AO4_IO_TIMEOUT_INFINITE is used, then the driver will wait indefinitely rather than timing out. The default is 10 seconds.

4.7.71. HSDI4AO4_IOCTL_WAIT_CANCEL

This service resumes all threads blocked via HSDI4AO4_IOCTL_WAIT_EVENT IOCTL calls (section 4.7.72, page 50), 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	HSDI4AO4_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 4.7.72.2 on page 51.
gsc	This specifies the set of HSDI4AO4_WAIT_GSC_* events whose wait requests are to be cancelled. Refer to section 4.7.72.3 on page 51.
alt	This is unused by the 16HSDI4AO4 driver and should be zero.
io	This specifies the set of HSDI4AO4_WAIT_IO_* events whose wait requests are to be cancelled. Refer to section 4.7.72.4 on page 52.
timeout_ms	This is unused by wait cancel operations.
count	Upon return this indicates the number of waits that were cancelled.

4.7.72. HSDI4AO4_IOCTL_WAIT_EVENT

This service blocks a thread until any one of a specified set of events occurs, or until a timeout lapses, whichever occurs first. The set of possible events to wait for are specified in the structure's main, gsc, alt and io fields. All field values must be valid and at least one event must be specified. If the thread is resumed because one of the referenced events has occurred, then the bit for the respective event is the only event bit that will be set. All other event bits and fields will be zero. (Multiple event bits will be set only if the events occur simultaneously.)

NOTE: The service waits only for the first of the specified events, not for all specified events.

NOTE: A wait timeout is reported via the gsc_wait_t structure's flags field having the GSC_WAIT_FLAG_TIMEOUT flag set, rather than via an ETIMEDOUT error.

Usage

Argument	Description
request	HSDI4AO4_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 4.7.72.1on page 51.
main	This specifies any number of GSC_WAIT_MAIN_* events that the thread is to wait for. Refer to section 4.7.72.2 on page 51.
gsc	This is unused by the 16HSDI4AO4 driver and must be zero.
alt	This is unused by the 16HSDI4AO4 driver and must be zero.
io	This specifies any number of HSDI4AO4_WAIT_IO_* events that the thread is to wait for. Refer to section 4.7.72.4 on page 52.

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.

4.7.72.1. gsc_wait_t.flags Options

Upon return from a wait request the wait structure's flags field will indicate the reason that the thread was resumed. Only one of the below options will be set.

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

4.7.72.2. 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 16HSDI4AO4 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 16HSDI4AO4.
GSC_WAIT_MAIN_PCI	This refers to any interrupt generated by the 16HSDI4AO4.
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.

4.7.72.3. gsc_wait_t.gsc Options

The wait structure's gsc field may specify any of the below firmware interrupt options.

Value	Description
-1	Retrieve the current setting.
HSDI4AO4_WAIT_GSC_AI_CLOCK	This refers to the analog input clock.
HSDI4AO4_WAIT_GSC_AI_BURST_DONE	This refers to the completion of an analog input burst.
HSDI4AO4_WAIT_GSC_AI_BURST_START	This refers to the beginning of an analog input burst.
HSDI4AO4_WAIT_GSC_AI_FAULT	This refers to an input buffer overflow or underflow.
HSDI4AO4_WAIT_GSC_AI_THRESH_H2L	This refers to a low to high transition of the analog input threshold level status.
HSDI4AO4_WAIT_GSC_AI_THRESH_L2H	This refers to a high to low transition of the analog input threshold level status.
HSDI4AO4_WAIT_GSC_AO_BURST_READY	This refers to readiness of the board to begin an analog output burst.
HSDI4AO4_WAIT_GSC_AO_CLOCK	This refers to the analog output clock.
HSDI4AO4_WAIT_GSC_AO_FAULT	This refers to an analog output buffer underflow, or to a data or frame overflow.
HSDI4AO4_WAIT_GSC_AO_LOAD_RDY_HL	This refers to a high to low transition of the output buffer's Load Ready status.
HSDI4AO4_WAIT_GSC_AO_LOAD_RDY_LH	This refers to a low to high transition of the output buffer's Load Ready status.

HSDI4AO4_WAIT_GSC_AO_THRESH_H2L	This refers to a high to low transition of the output buffer's threshold level status.
HSDI4AO4_WAIT_GSC_AO_THRESH_L2H	This refers to a low to high transition of the output buffer's threshold level status.
HSDI4AO4_WAIT_GSC_AUTOCAL_DONE	This refers to the completion of an autocalibration cycle.
HSDI4AO4_WAIT_GSC_DIO_0_L2H	This refers to a low to high transition at digital I/O signal D0.

4.7.72.4. gsc_wait_t.io Options

The wait structure's io field may specify any of the below event options. These events are generated in response to application board data read requests.

Fields	Description
HSDI4AO4_WAIT_IO_RX_ABORT	This refers to read requests which have been aborted.
HSDI4AO4_WAIT_IO_RX_DONE	This refers to read requests which have been satisfied.
HSDI4AO4_WAIT_IO_RX_ERROR	This refers to read requests which end due to an error.
HSDI4AO4_WAIT_IO_RX_TIMEOUT	This refers to read requests which end due to the timeout period lapse.
HSDI4AO4_WAIT_IO_TX_ABORT	This refers to write requests which have been aborted.
HSDI4AO4_WAIT_IO_TX_DONE	This refers to write requests which have been satisfied.
HSDI4AO4_WAIT_IO_TX_ERROR	This refers to write requests which end due to an error.
HSDI4AO4_WAIT_IO_TX_TIMEOUT	This refers to write requests which end due to the timeout period lapse.

4.7.73. HSDI4AO4_IOCTL_WAIT_STATUS

This service count all threads blocked via the HSDI4AO4_IOCTL_WAIT_EVENT IOCTL service (section 4.7.72, page 50), according to the provided criteria. A match is made when a waiting thread's wait criteria matches any of the criteria specified in the structure passed to this service.

> **NOTE:** The driver itself makes use of the wait services for various internal operations. Driver initiated waits are ignored by application status requests.

Usage

Argument	Description
request	HSDI4AO4_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 4.7.72.2 on page 51.

gsc	This specifies the set of HSDI4AO4_WAIT_GSC_* events whose wait requests are to be counted. Refer to section 4.7.72.3 on page 51.
alt	This is unused by the 16HSDI4AO4 driver and should be zero.
io	This specifies the set of HSDI4AO4_WAIT_IO_* events whose wait requests are to be counted. Refer to section 4.7.72.4 on page 52.
timeout_ms	This is unused by wait status operations.
count	Upon return this indicates the number of waits that met any of the specified criteria.

5. The Driver

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

5.1. Files

The device driver files are summarized in the table below.

Description	Files	Location
Source Files	`*.c`, `*.h` …	…/driver/
Header File	`16hsdi4ao4.h`	
Driver File	`16hsdi4ao4.ko` † `16hsdi4ao4.o` ‡	

† This is for kernel versions 2.6 and later.
‡ This is for kernel versions 2.4 are earlier.

5.2. Build

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

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

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

```
make clean
```

3. Build the driver by issuing the below command.

```
make
```

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

5.3. Startup

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

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

5.3.1. Manual Driver Startup Procedures

Start the driver manually by following the below listed steps.

NOTE: The following steps may require elevated privileges.

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

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

   ```
   ./start
   ```

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

   NOTE: The 16HSDI4AO4 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 16hsdi4ao4 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/16hsdi4ao4.*
   ```

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/16hsdi4ao4/driver/start
   ```

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

2. Load the driver and create the required device nodes by rebooting the system.

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

5.3.2.1. File rc.local Not Present

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

```
#!/bin/bash

# Add your local content here.
```

5.3.2.2. Default rc.local File Permissions

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

5.3.2.3. systemd Installations

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

```
systemctl enable rc-local
```

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

5.3.2.4. systemd and rc.local Timing

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

5.3.2.5. SElinux Implications

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

```
yum install setroubleshoot setools
```

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

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

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

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

5.4. Verification

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

1. Verify that the file `/proc/16hsdi4ao4` 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/16hsdi4ao4
```

5.5. Version

The driver version number can be obtained in a variety of ways. It is reported by the driver both when the driver is loaded and when it is unloaded (depending on kernel configuration options, this may be visible only in places such as `/var/log/messages`). It is reported in the text file `/proc/16hsdi4ao4` while the driver is loaded and running. The version number is also given in the file `release.txt` in the root install directory.

5.6. Shutdown

Shutdown the driver following the below listed steps.

> **NOTE**: The following steps may require elevated privileges.

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

```
rmmod 16hsdi4ao4
```

2. Verify that the driver module has been unloaded by issuing the below command. The module name `16hsdi4ao4` should not be in the listed output.

```
lsmod
```

6. Document Source Code Examples

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

6.1. Files

The library files are summarized in the table below.

Description	Files	Location
Source Files	`*.c`, `*.h` …	…`/docsrc/`
Header File	`16hsdi4ao4_dsl.h`	…`/include/`
Library File	`16hsdi4ao4_dsl.a`	…`/lib/`

6.2. Build

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

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

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

```
make clean
```

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

```
make
```

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

6.3. Library Use

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

7. Utilities Source Code

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

For each API function there is a corresponding utility source file with a corresponding utility service. As an example, for the API function `hsdi4ao4_open()` there is the utility file `open.c` containing the utility function `hsdi4ao4_open_util()`. The naming pattern is as follows: API function `hsdi4ao4_xxxx()`, utility file name `xxxx.c`, utility function `hsdi4ao4_xxxx_util()`. Additionally, for each IOCTL code there is a corresponding utility source file with a corresponding utility service. As an example, for IOCTL code `HSDI4AO4_IOCTL_QUERY` there is the utility file `util_query.c` containing the utility function `hsdi4ao4_query()`. The naming pattern is as follows: IOCTL code `HSDI4AO4_IOCTL_XXXX`, utility file name `util_xxxx.c`, utility function `hsdi4ao4_xxxx()`.

7.1. Files

The utility files are summarized in the table below.

Description	Files	Location
Source Files	`*.c`, `*.h` …	…`/utils/`
Header File	`16hsdi4ao4_utils.h`	…`/include/`

Library Files	`16hsdi4ao4_utils.a`
	`gsc_utils.a`
	`os_utils.a`
	`plx_utils.a`
		…`/lib/`

7.2. Build

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

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

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

   ```
   make clean
   ```

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

   ```
   make
   ```

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

7.3. Library Use

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

8. Operating Information

This section explains some basic operational procedures for using the 16HSDI4AO4. This is in no way intended to be a comprehensive guide. This is simply to address a very few issues relating to their use.

8.1. Debugging Aids

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

8.1.1. Device Identification

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

Description	File	Location
Application	id	…/id/

8.1.2. Detailed Register Dump

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

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

Description	File/Name	Location
Function	hsdi4ao4_reg_list()	Source File
Source File	util_reg.c	…/utils/
Header File	16hsdi4ao4_utils.h	…/include/
Library File	16hsdi4ao4_utils.a	…/lib/

8.2. Analog Input Configuration

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

Item	Name/File	Location
Function	hsdi4ao4_config_ai()	Source File
Source File	util_config_ai.c	…/utils/
Header File	16hsdi4ao4_utils.a	…/include/
Library File	16hsdi4ao4_utils.a	…/lib/

8.3. Analog Output Configuration

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

Item	Name/File	Location
Function	`hsdi4ao4_config_ao()`	Source File
Source File	`util_config_ao.c`	…`/utils/`
Header File	`16hsdi4ao4_utils.h`	…`/include/`
Library File	`16hsdi4ao4_utils.a`	…`/lib/`

8.4. Data Transfer Modes

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

PIO - Programmed I/O

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

8.4.1. BMDMA - Block Mode DMA

This mode is intended for data transfers that do not exceed the size of the 16HSDI4AO4 input/output buffer. Here, the board's DMA engine is used to perform a hardware-controlled transfer which does not require processer intervention to move the data. In this mode the DMA read transfer is initiated only when the input buffer contains sufficient data to fulfill the request. For write requests a DMA transfer is initiated only when the output buffer contains sufficient space to fulfill the request. This is a very efficient I/O method. However, for small requests PIO is more efficient.

8.4.2. DMDMA - Demand Mode DMA

This DMA transfer mode is similar to the block mode, except that a transfer for the entire amount of data is initiated immediately and is not limited to the size of the board's FIFO. Here however, the actual movement of data occurs as the data becomes available in the input buffer, for reads, or as space becomes available in the output buffer, for writes. This is the most efficient method supported. However, for small requests PIO is more efficient.

9. Sample Applications

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

9.1. aout - Analog Output - …/aout/

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

9.2. din - Digital Input - …/din/

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

9.3. dout - Digital Output - …/dout/

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

9.4. 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 input sample rate then reads 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. sbtest - Single Board Test - …/sbtest/

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

9.10. signals - Digital Signals - …/signals/

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

9.11. txrate - Transmit Rate - …/txrate/

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

Document History

Revision	Description
July 24, 2023	Updated to version 2.8.104.47.0. Numerous, minor editorial changes. Updated the description of the Input and Output Buffer Clear services. Updated the description of the Autocalibration service. Expanded the Query IOCTL service options. Changed spelling of all AUTO_CAL content to AUTOCAL (case insensitive).
April 21, 2023	Updated to version 2.7.102.45.1.
March 8, 2023	Updated to version 2.7.102.45.0. Updated the kernel support table. Added section on environment variables. Updated the information for the open and close calls. Minor editorial changes.
October 4, 2021	Updated to version 2.6.94.37.0. Updated the kernel support table. Minor editorial changes. Added a licensing subsection. Added WAIT_EVENT note. Expanded automatic startup information.
June 12, 2019	Updated to version 2.5.86.28.0. Updated the kernel support table. Minor editorial changes. Updated the software architecture documentation.
March 6, 2019	Updated to version 2.4.82.26.0. Updated Block Mode DMA macro and associated information. Minor editorial changes. Document reorganization.
July 12, 2018	Updated to version 2.3.79.25.0.
July 10, 2018	Updated to version 2.2.79.25.0. Updated the inside cover page. Updated the CPU and kernel support section. Minor editorial changes.
December 1, 2016	Updated to version 2.1.68.18.0. Removed the `built` field from the `/proc/` file. Updated the kernel support table. Updated the command line arguments for the `din` and `signals` sample application. Organized sample applications alphabetically. Updated the usage of the Wait Event `timeout_ms` field. Updated material on the open call. Added open access mode descriptions. Updated the kernel support table. Added support for infinite I/O timeouts. Added a section for general operating information. Made various miscellaneous updates. Some document reorganization.
September 14, 2015	Updated to version 2.0.60.8.0. Updated the device node name to include a period before the device index. Removed double underscore that prefaced various data types.
February 27, 2014	Updated to version 1.5.52.0. Updated the kernel support table.
January 9, 2014	Updated to version 1.4.51.0. Updated the kernel support table.
November 15, 2013	Updated to version 1.4.50.0.
July 16, 2013	Updated to version 1.4.45.0. Updated the kernel support table.
July 19, 2012	Updated to version 1.4.39.0. Updated the kernel support table.
January 13, 2012	Updated to version 1.3.35.0.
November 16, 2011	Updated to version 1.2.32.0.
February 2, 2011	Updated to version 1.1.22.0. Various editorial changes. Updated the CPU and Kernel Support information. Updated the comments for the Initialize IOCTL service. Changed the spelling of various Autocalibration related software items.
July 9, 2010	Updated to version 1.0.17.0.
September 22, 2009	Updated to version BETA.0.10.0. Numerous updates and additions.
June 26, 2009	Initial release, version BETA.0.8.0.