20AO8C500K

20-bit, 8 channel, 500K S/S/Ch Analog Output

PCIe-20AO8C500K

Linux Device Driver And API Library User Manual

Manual Revision: September 21, 2023
Driver Release Version 1.0.105.47.0

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 20AO8C500K 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 20AO8C500K hardware. The API Library and driver interfaces are based on the board's functionality.

1.2. Acronyms

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

Acronyms	Description
API	Application Programming Interface
BMDMA	Block Mode DMA
DAC	Digital-to-Analog Converter
DIO	Digital I/O
DMA	Direct Memory Access
DMDMA	Demand Mode DMA
GSC	General Standards Corporation
PCI	Peripheral Component Interconnect
PCIe	PCI Express
PIO	Programmed I/O

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 20AO8C500K installation directory or any of its subdirectories.
20AO8C500K	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 20AO8C500K hardware.
Application	This is a user mode process, which runs in user space with user mode privileges.
Driver	This is the 20AO8C500K 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 20AO8C500K applications. The overall architecture is illustrated in Figure 1 below.

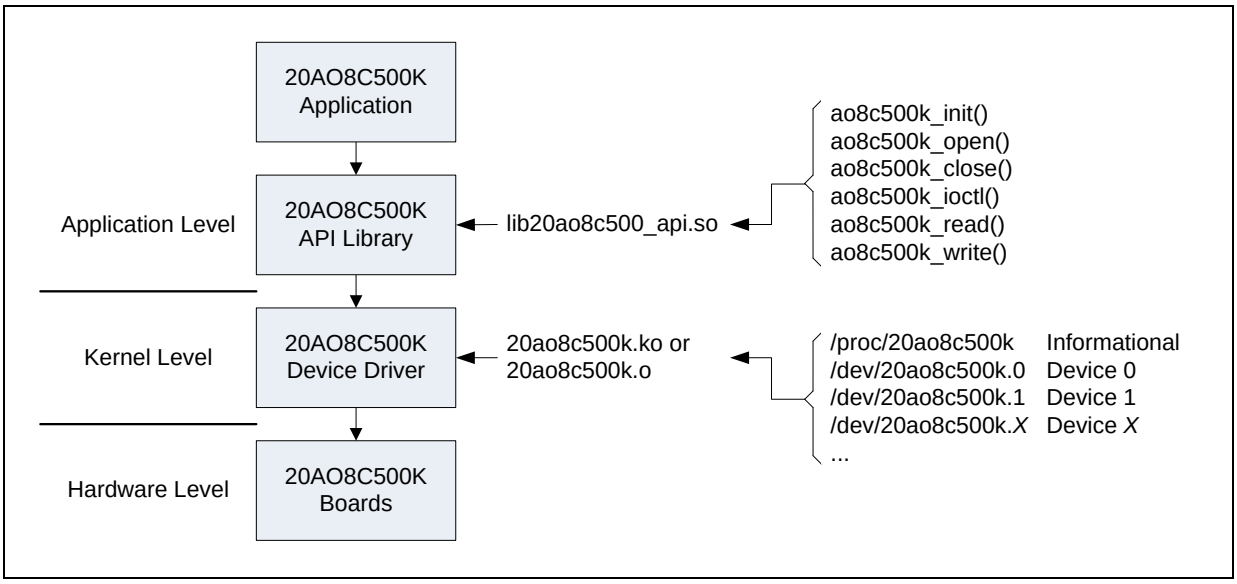

Figure 1 Basic architectural representation.

1.4.2. API Library

The primary means of accessing 20AO8C500K boards is via the 20AO8C500K API Library. This library forms a thin 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 20AO8C500K 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 20AO8C500K is a high-performance, 18-bit analog output board that incorporates 8 or 4 output channels. The host side connection is PCI based and the form factor is according to the model ordered. The board is capable of outputting data at up to 500K samples per second over each channel. Internal clocking permits sampling rates from 500K samples per second down to two samples per second. Onboard storage permits data buffering of up to 256K samples, for all channels collectively, between the PCI bus and the cable interface. This allows the 20AO8C500K to sustain continuous throughput to the cable interface independent of the PCI bus interface. The 20AO8C500K also permits multiple boards to be synchronized so that all boards output data in unison. The board also incorporates eight bidirectional digital I/O lines.

1.6. Reference Material

The following reference material may be of particular benefit in using the 20AO8C500K. The specifications provide the information necessary for an in depth understanding of the specialized features implemented on this board.

- The applicable *20AO8C500K 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/20ao8c500k` file will be "`no`".

2.2. The /proc/ File System

While the driver is running, the text file `/proc/20ao8c500k` 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: 1.0.105.47
32-bit support: yes
boards: 1
models: 20AO8C500K
```

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
`20ao8c500k.linux.tar.gz`	This archive contains the driver, the API Library and all related files.
`20ao8c500k_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
`20ao8c500k/`	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 52).
`…/driver/`	This directory contains the device driver source files (section 5, page 48).
`…/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 57).
…/utils/	This directory contains the source files for the utility libraries used by the sample applications (section 7, page 53).

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 `20ao8c500k.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 `20ao8c500k` in the current directory, and then copies all of the archive's files into this new directory.

```
tar –xzvf 20ao8c500k.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 51).

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 20ao8c500k.linux.tar.gz 20ao8c500k
```

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

```
rm -f /dev/20ao8c500k.*
```

5. If the automatic startup procedure was adopted (section 5.3.2, page 49), then edit the system startup script `rc.local` and remove the line that invokes the 20AO8C500K'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 (…`/20ao8c500k/`).

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/lib20ao8c500k_api.so
Defined and Not Empty	==== Linking: ../lib/lib20ao8c500k_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/20ao8c500k_utils.a`
Defined and Not Empty	`==== Linking: ../lib/20ao8c500k_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 20AO8C500K 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 20AO8C500K 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 20AO8C500K specific header files. Therefore, sources may include only this one 20AO8C500K header and make files may reference only this one 20AO8C500K include directory.

Description	File	Location
Header File	`20ao8c500k_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 20AO8C500K 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 20AO8C500K static library and only this one 20AO8C500K library directory.

Description	File	Location
Static Library	`20ao8c500k_main.a`	`…/lib/`
	`20ao8c500k_multi.a`	

> **NOTE**: For applications using the 20AO8C500K and no other GSC devices, link the `20ao8c500k_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 20AO8C500K API Library is implemented as a shared library and is thus not linked with the 20AO8C500K Main Library. The API Library must be linked with applications by adding the argument `-l20ao8c500k_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 20AO8C500K API Library is the software interface between user applications and the 20AO8C500K device driver. The interface is accessed by including the header file `20ao8c500k_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	`20ao8c500k_api.h`	`…/include/`
Library File	`lib20ao8c500k_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 API Library shared library is copied to `/usr/lib/`. Therefore, these steps may require elevated privileges.

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

2. Remove existing build targets using the below command.

   ```
   make clean
   ```

3. Compile the source files and build the library by issuing the below command. 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	20ao8c500k_api.h	…/include/	
Shared Object Library	lib20ao8c500k_api.so	…/lib/	
		/usr/lib/	-l20ao8c500k_api

4.4. Macros

The API Library and driver interfaces include the following macros, which are defined in 20ao8c500k.h.

4.4.1. IOCTL Services

The IOCTL macros are documented in section 4.7 beginning on page 25.

4.4.2. Registers

The following gives the complete set of 20AO8C500K registers.

4.4.2.1. GSC Registers

The following table gives the complete set of GSC specific 20AO8C500K 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 *20AO8C500K 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
AO8C500K_GSC_ACR	Assembly Configuration Register (ACR)
AO8C500K_GSC_AOBR	Analog Output Buffer Register (AOBR)
AO8C500K_GSC_ASIOCR	Auxiliary Sync I/O Register (ASIOR)
AO8C500K_GSC_AVR	Autocal Values Register (AVR)
AO8C500K_GSC_BCR	Board Control Register (BCR)
AO8C500K_GSC_BOOR	Buffered Output Operations Register (BOOR)
AO8C500K_GSC_DIOPR	Digital I/O Port Register (DIOPR)
AO8C500K_GSC_OBSR	Output Buffer Size Register (OBSR)
AO8C500K_GSC_OBTR	Output Buffer Threshold Register (OBTR)
AO8C500K_GSC_OCR	Output Configuration Register (OCR)
AO8C500K_GSC_PSR	Primary Status Register (PSR)
AO8C500K_GSC_RAGR	Rate-A Generator Register (RAGR)
AO8C500K_GSC_RBGR	Rate-B Generator Register (RBGR)

4.4.2.2. PCI Configuration Registers

Access to the PCI registers is seldom required so these registers are not listed here. For the complete list of the PCI register identifiers refer to the driver header file gsc_pci9056.h, which is automatically included via 20ao8c500k_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 20ao8c500k_api.h.

4.5. Data Types

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

4.6. Functions

The interface includes the following functions. The return values reflect the completion status of the requested operation. A 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. ao8c500k_close()

This function is the entry point to close a connection to an open 20AO8C500K board. The board is put in an initialized state before this call returns.

Prototype

```
int ao8c500k_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 "20ao8c500k_dsl.h"

int ao8c500k_close_dsl(int fd)
{
    int errs;
    int ret;

    ret = ao8c500k_close(fd);

    if (ret)
        printf("ERROR: ao8c500k_close() returned %d\n", ret);

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

4.6.2. ao8c500k_init()

This function is the entry point to initializing the 20AO8C500K 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 ao8c500k_init(void);
```

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

Example

```
#include <stdio.h>

#include "20ao8c500k_dsl.h"

int ao8c500k_init_dsl(void)
{
    int errs;
    int ret;

    ret = ao8c500k_init();

    if (ret)
        printf("ERROR: ao8c500k_init() returned %d\n", ret);

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

4.6.3. ao8c500k_ioctl()

This function is the entry point to performing setup and control operations on a 20AO8C500K. This function should only be called after a successful open of the respective device. The specific operation performed varies according to the `request` argument. The `request` argument also governs the use and interpretation of the `arg` argument. The set of supported options for the `request` argument consists of the IOCTL services supported by the driver, which are defined in section 4.7 beginning on page 25.

Prototype

```
int ao8c500k_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 "20ao8c500k_dsl.h"

int ao8c500k_ioctl_dsl(int fd, int request, void* arg)
{
    int errs;
    int ret;

    ret = ao8c500k_ioctl(fd, request, arg);

    if (ret)
        printf("ERROR: ao8c500k_ioctl() returned %d\n", ret);

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

4.6.4. ao8c500k_open()

This function is the entry point to open a connection to a 20AO8C500K board. Before returning, the initialize IOCTL service is called to reset all hardware and software settings to their defaults.

Prototype

```
int ao8c500k_open(int device, int share, int* fd);
```

Argument	Description
device	This is the zero-based index of the 20AO8C500K to access. †
share	Open the device in Shared Access Mode? If non-zero the device is opened in Shared Access Mode (see below). If zero the device is opened in Exclusive Access Mode (see below).
fd	The device handle is returned here. The pointer cannot be NULL. Values returned are as follows. **Value** / **Description** >= 0 — This is the handle to use to access the device in subsequent calls. -1 — There was an error. The device is not accessible.

† If the index value is -1, then the API Library accesses /proc/20ao8c500k.

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

Example

```
#include <stdio.h>

#include "20ao8c500k_dsl.h"
```

```
int ao8c500k_open_dsl(int device, int share, int* fd)
{
    int errs;
    int ret;

    ret = ao8c500k_open(device, share, fd);

    if (ret)
        printf("ERROR: ao8c500k_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. ao8c500k_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).

> **NOTE:** The read service has no functionality for reading from 20AO8C500K devices. Attempts to read from 20AO8C500K devices will return an error.

Prototype

```
int ao8c500k_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 will be put here.
`bytes`	This is the desired number of bytes to read.

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

Example

```
#include <stdio.h>

#include "20ao8c500k_dsl.h"

int ao8c500k_read_dsl(int fd, void* dst, size_t bytes, size_t* qty)
{
    int errs;
    int ret;

    ret = ao8c500k_read(fd, dst, bytes);

    if (ret < 0)
        printf("ERROR: ao8c500k_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. ao8c500k_write()

This function is the entry point to writing data to an open 20AO8C500K. The function writes up to bytes bytes to the device.

> **NOTE:** Applications may experience improved responsiveness with read requests by coordinating the Buffer Size setting with the number of samples to write. Refer to the AO8C500K_IOCTL_BUF_FILL_LEVEL service of section 4.7.10 on page 28.

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

Prototype

```
int ao8c500k_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 4.7.49, page 42) before the entire request could be satisfied.
< 0	An error occurred. See error value description above.

Example

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

```
#include "20ao8c500k_dsl.h"

int ao8c500k_write_dsl(int fd, const void* src, size_t bytes, size_t*
qty)
{
    int errs;
    int ret;

    ret = ao8c500k_write(fd, src, bytes);

    if (ret < 0)
        printf("ERROR: ao8c500k_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 20AO8C500K API Library and device driver implement the following IOCTL services. Each service is described along with the applicable `ao8c500k_ioctl()` function arguments.

4.7.1. AO8C500K_IOCTL_AO_CONFIG

This service configures the analog output wiring configuration.

Usage

Argument	Description
request	AO8C500K_IOCTL_AO_CONFIG
arg	s32*

Valid argument values are as follows.

Value	Description
-1	Retrieve the current setting.
AO8C500K_AO_CONFIG_DIFF	Select differential analog output configuration.
AO8C500K_AO_CONFIG_SE	Select single ended analog output configuration.

4.7.2. AO8C500K_IOCTL_AUTOCAL

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

> **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	AO8C500K_IOCTL_AUTOCAL
arg	Not used.

4.7.3. AO8C500K_IOCTL_AUTOCAL_STATUS

This service retrieves the autocalibration completion status.

Usage

Argument	Description
request	AO8C500K_IOCTL_AUTOCAL_STATUS
arg	s32*

The value returned will be one of the following.

Value	Description
AO8C500K_AUTOCAL_STATUS_ACTIVE	Autocalibration is in progress.
AO8C500K_AUTOCAL_STATUS_FAIL	Autocalibration failed.
AO8C500K_AUTOCAL_STATUS_PASS	Autocalibration passed.

4.7.4. AO8C500K_IOCTL_AUX_CLOCK_MODE

This service configures the operating mode for the Auxiliary Clock signal.

Usage

Argument	Description
request	AO8C500K_IOCTL_AUX_CLOCK_MODE
arg	s32*

Valid argument values are as follows.

Value	Description
-1	Retrieve the current setting.
AO8C500K_AUX_MODE_DISABLE	This disables the Auxiliary Clock signal.
AO8C500K_AUX_MODE_INPUT	This configures the Auxiliary Clock signal as an input.
AO8C500K_AUX_MODE_OUTPUT	This configures the Auxiliary Clock signal as an output.

4.7.5. AO8C500K_IOCTL_AUX_IN_POL

This service configures the polarity of the auxiliary input signals.

Usage

Argument	Description
request	AO8C500K_IOCTL_AUX_IN_POL
arg	s32*

Valid argument values are as follows.

Value	Description
-1	Retrieve the current setting.
AO8C500K_AUX_POL_H2L	This configures the signals to detect high-to-low input transitions.
AO8C500K_AUX_POL_L2H	This configures the signals to detect low-to-high input transitions.

4.7.6. AO8C500K_IOCTL_AUX_NOISE

This service configures the noise immunity level for the auxiliary cable signals.

Usage

Argument	Description
request	AO8C500K_IOCTL_AUX_NOISE
arg	s32*

Valid argument values are as follows.

Value	Description
-1	Retrieve the current setting.
AO8C500K_AUX_NOISE_HIGH	This configures the signals for high noise operation.
AO8C500K_AUX_NOISE_LOW	This configures the signals for low noise operation.

4.7.7. AO8C500K_IOCTL_AUX_OUT_POL

This service configures the polarity of the auxiliary output signals.

Usage

Argument	Description
request	AO8C500K_IOCTL_AUX_OUT_POL
arg	s32*

Valid argument values are as follows.

Value	Description
-1	Retrieve the current setting.
AO8C500K_AUX_POL_L2H	This configures the output signals to produce low to high going pulses.
AO8C500K_AUX_POL_H2L	This configures the output signals to produce high to low going pulses.

4.7.8. AO8C500K_IOCTL_AUX_TRIGGER_MODE

This service configures the operating mode for the Auxiliary Sync signal.

Usage

Argument	Description
request	AO8C500K_IOCTL_AUX_TRIGGER_MODE
arg	s32*

Valid argument values are as follows.

Value	Description
-1	Retrieve the current setting.
AO8C500K_AUX_MODE_DISABLE	This disables the Auxiliary Sync signal.

AO8C500K_AUX_MODE_INPUT	This configures the Auxiliary Sync signal as an input.
AO8C500K_AUX_MODE_OUTPUT	This configures the Auxiliary Sync signal as an output.

4.7.9. AO8C500K_IOCTL_BUF_CLEAR

This service immediately clears the current content from the output buffer. The service also clears the associated data overflow and frame overflow status bits.

Usage

Argument	Description
request	AO8C500K_IOCTL_BUF_CLEAR
arg	Not used.

4.7.10. AO8C500K_IOCTL_BUF_FILL_LEVEL

This service reports the output buffer fill level.

Usage

Argument	Description
request	AO8C500K_IOCTL_BUF_FILL_LEVEL
arg	s32*

The service returns values in the range from zero to 256K.

4.7.11. AO8C500K_IOCTL_BUF_MODE

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

Usage

Argument	Description
request	AO8C500K_IOCTL_BUF_MODE
arg	s32*

Valid argument values are as follows.

Value	Description
-1	Retrieve the current setting.
AO8C500K_BUF_MODE_CIRC	Buffer data is recycled when it exits the buffer.
AO8C500K_BUF_MODE_OPEN	The buffer data is not recycled when it exits the buffer.

4.7.12. AO8C500K_IOCTL_BUF_OVER_DATA

This service operates on the Buffer Overflow status.

Usage

Argument	Description
request	AO8C500K_IOCTL_BUF_OVER_DATA
arg	s32*

Valid argument values are as follows.

Value	Description
AO8C500K_BUF_OVER_DATA_CHK	Report if an overflow has occurred.
AO8C500K_BUF_OVER_DATA_CLR	Clear the overflow status.

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

Value	Description
AO8C500K_BUF_OVER_DATA_NO	An overflow did not occur.
AO8C500K_BUF_OVER_DATA_YES	An overflow did occur.

4.7.13. AO8C500K_IOCTL_BUF_OVER_FRAME

This service operates on the Frame Overflow status.

Usage

Argument	Description
request	AO8C500K_IOCTL_BUF_OVER_FRAME
arg	s32*

Valid argument values are as follows.

Value	Description
AO8C500K_BUF_OVER_FRAME_CHK	Report if an overflow has occurred.
AO8C500K_BUF_OVER_FRAME_CLR	Clear the overflow status.

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

Value	Description
AO8C500K_BUF_OVER_FRAME_NO	An overflow did not occur.
AO8C500K_BUF_OVER_FRAME_YES	An overflow did occur.

4.7.14. AO8C500K_IOCTL_BUF_STATUS

This service reports the output buffer fill level status.

Usage

Argument	Description
request	AO8C500K_IOCTL_BUF_STATUS
arg	s32*

The service returns one of the following values.

Value	Description
AO8C500K_BUF_STATUS_AE	The output buffer fill level is at or below the Threshold level, but it is not empty.
AO8C500K_BUF_STATUS_AF	The output buffer fill level is above the Threshold level, but it is not full.
AO8C500K_BUF_STATUS_EMPTY	The output buffer is empty.
AO8C500K_BUF_STATUS_FULL	The output buffer is full.

4.7.15. AO8C500K_IOCTL_BUF_THRESH_LEVEL

This service configures the fill level for the output buffer Threshold status. The status is asserted when the fill level exceeds the configured Threshold value.

Usage

Argument	Description
request	AO8C500K_IOCTL_BUF_THRESH_LEVEL
arg	s32*

Valid values are from zero to 256K (0x40000) and −1. The value of −1 will retrieve the current setting.

4.7.16. AO8C500K_IOCTL_BUF_THRESH_STATUS

This service reports the output buffer Threshold status.

Usage

Argument	Description
request	AO8C500K_IOCTL_BUF_THRESH_STATUS
arg	s32*

The service returns one of the following values.

Value	Description
AO8C500K_BUF_THRESH_STATUS_CLR	The fill level is at or below the Threshold level.
AO8C500K_BUF_THRESH_STATUS_SET	The fill level exceeds the Threshold level.

4.7.17. AO8C500K_IOCTL_BURST_ENABLE

This service enables or disables output bursting.

Usage

Argument	Description
request	AO8C500K_IOCTL_BURST_ENABLE
arg	s32*

Valid argument values are as follows.

Value	Description
-1	Retrieve the current state.
AO8C500K_BURST_ENABLE_NO	This refers to output bursting being disabled.
AO8C500K_BURST_ENABLE_YES	This refers to output bursting being enabled.

4.7.18. AO8C500K_IOCTL_BURST_READY

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

Usage

Argument	Description
request	AO8C500K_IOCTL_BURST_READY
arg	s32*

The service returns one of the following values.

Value	Description
AO8C500K_BURST_READY_NO	The board is not ready for burst initiation.
AO8C500K_BURST_READY_YES	The board is ready for burst initiation.

4.7.19. AO8C500K_IOCTL_BURST_TRIG_SRC

This service configures the trigger source for triggered burst operations.

Usage

Argument	Description
request	AO8C500K_IOCTL_BURST_TRIG_SRC
arg	s32*

Valid argument values are as follows.

Value	Description
-1	Retrieve the current setting.
AO8C500K_BURST_TRIG_SRC_EXT	Utilize external burst triggering.
AO8C500K_BURST_TRIG_SRC_RBG	Utilize the Rate-B Generator output.

4.7.20. AO8C500K_IOCTL_CHANNEL_SEL

This service enables or disables channels for outputting data.

Usage

Argument	Description
request	AO8C500K_IOCTL_CHANNEL_SEL
arg	s32*

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

4.7.21. AO8C500K_IOCTL_CLOCK_ENABLE

This service enables and disables clocking of output data.

Usage

Argument	Description
request	AO8C500K_IOCTL_CLOCK_ENABLE
arg	s32*

Valid argument values are as follows.

Value	Description
-1	Retrieve the current setting.
AO8C500K_CLOCK_ENABLE_NO	This disables the output sample clock.
AO8C500K_CLOCK_ENABLE_YES	This enables the output sample clock.

4.7.22. AO8C500K_IOCTL_CLOCK_READY

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

Usage

Argument	Description
request	AO8C500K_IOCTL_CLOCK_READY
arg	s32*

The service returns one of the following values.

Value	Description
AO8C500K_CLOCK_READY_NO	The board is not ready for a clock.
AO8C500K_CLOCK_READY_YES	The board is ready for a clock.

4.7.23. AO8C500K_IOCTL_CLOCK_SRC

This service configures the source for the output sample clock.

Usage

Argument	Description
request	AO8C500K_IOCTL_CLOCK_SRC
arg	s32*

Valid argument values are as follows.

Value	Description
-1	Retrieve the current setting.
AO8C500K_CLOCK_SRC_EXT	This selects the external clock input as the clock source.
AO8C500K_CLOCK_SRC_RAG	This selects the Rate-A Generator as the source.

4.7.24. AO8C500K_IOCTL_CLOCK_SW

This service initiates an output clock cycle. The service waits for up to the write timeout period for the operation to complete (though not the infinite option). (See AO8C500K_IOCTL_TX_IO_TIMEOUT, section 4.7.49, page 42.)

Usage

Argument	Description
request	AO8C500K_IOCTL_CLOCK_SW
arg	Not used.

4.7.25. AO8C500K_IOCTL_DATA_FORMAT

This service sets the data encoding format.

Usage

Argument	Description
request	AO8C500K_IOCTL_DATA_FORMAT
arg	s32*

Valid argument values are as follows.

Value	Description
-1	Retrieve the current setting.
AO8C500K_DATA_FORMAT_2S_COMP	Select the Twos Compliment data format.
AO8C500K_DATA_FORMAT_OFF_BIN	Select the Offset Binary encoding format.

4.7.26. AO8C500K_IOCTL_DIO_DIR_OUT

This service configures the direction of the digital I/O signals, which are organized into two nibble-wide ports.

Usage

Argument	Description
request	AO8C500K_IOCTL_DIO_DIR_OUT
arg	s32*

Valid argument values are as follows.

Value	Description
-1	Retrieve the current setting.
AO8C500K_DIO_DIR_OUT_D3_D0	DIO lines D3 through D0 are output and the others are input.
AO8C500K_DIO_DIR_OUT_D7_D0	DIO lines D7 through D0 are output.
AO8C500K_DIO_DIR_OUT_D7_D4	DIO lines D7 through D4 are output and the others are input.
AO8C500K_DIO_DIR_OUT_NONE	All DIO lines are inputs.

4.7.27. AO8C500K_IOCTL_DIO_READ

This service reads the value of the eight digital I/O lines.

Usage

Argument	Description
request	AO8C500K_IOCTL_DIO_READ
arg	s32*

The value returned is from 0x00 through 0xFF. The bit values read are the levels at the cable interface for those bits configured as inputs and the last value written for those configured as outputs.

4.7.28. AO8C500K_IOCTL_DIO_WRITE

This service writes a value to the eight digital I/O lines.

Usage

Argument	Description
request	AO8C500K_IOCTL_DIO_WRITE
arg	s32*

Valid argument values are 0x00 through 0xFF. All bit values are latched and appear at the cable interface only when configured as outputs.

4.7.29. AO8C500K_IOCTL_GEN_A_ENABLE

This service enables or disables the Rate-A Generator.

Usage

Argument	Description
request	AO8C500K_IOCTL_GEN_A_ENABLE
arg	s32*

Valid argument values are as follows.

Value	Description
-1	Retrieve the current setting.
AO8C500K_GEN_ENABLE_NO	This disables the rate generator.
AO8C500K_GEN_ENABLE_YES	This enables the rate generator.

4.7.30. AO8C500K_IOCTL_GEN_A_NDIV

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

Usage

Argument	Description
request	AO8C500K_IOCTL_GEN_A_NDIV
arg	s32*

Valid argument values are in the range from 90 to 0xFFFFFF, and -1. The value -1 is used to retrieve the current setting.

4.7.31. AO8C500K_IOCTL_GEN_B_ENABLE

This service enables or disables the Rate-B Generator.

Usage

Argument	Description
request	AO8C500K_IOCTL_GEN_B_ENABLE
arg	s32*

Valid argument values are as follows.

Value	Description
-1	Retrieve the current setting.
AO8C500K_GEN_ENABLE_NO	This disables the rate generator.
AO8C500K_GEN_ENABLE_YES	This enables the rate generator.

4.7.32. AO8C500K_IOCTL_GEN_B_NDIV

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

Usage

Argument	Description
request	AO8C500K_IOCTL_GEN_B_NDIV
arg	s32*

Valid argument values are in the range from 90 to 0xFFFFFF, and -1. The value -1 is used to retrieve the current setting.

4.7.33. AO8C500K_IOCTL_INITIALIZE

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

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

Usage

Argument	Description
request	AO8C500K_IOCTL_INITIALIZE
arg	Not used.

4.7.34. AO8C500K_IOCTL_IRQ_ENABLE

This service enables and disabled firmware interrupts. Interrupts referenced in the request are enabled. All others are disabled.

Usage

Argument	Description
request	AO8C500K_IOCTL_IRQ_ENABLE
arg	s32*

Valid argument values include any combination of the options listed in the table below, and -1. The value -1 is used to retrieve the current setting.

Value	Description
-1	Retrieve the current setting.
AO8C500K_IRQ_AUTOCAL_DONE	This refers to the completion of autocalibration.
AO8C500K_IRQ_BUF_ERROR	This refers to the occurrence of a Buffer Overflow or of a Frame Overflow.
AO8C500K_IRQ_BUF_THR_H2L	This refers to the buffer fill level dropping to the Threshold level.

AO8C500K_IRQ_BUF_THR_L2H	This refers to the buffer fill level rising above the Threshold level.
AO8C500K_IRQ_BURST_READY	This refers to the board becoming ready for a burst trigger.
AO8C500K_IRQ_DIO0_L2H	This refers to a low to high transition at the D0 digital I/O line.
AO8C500K_IRQ_LOAD_READY_H2L	This refers to a circular buffer becoming not ready to receive data.
AO8C500K_IRQ_LOAD_READY_L2H	This refers to a circular buffer becoming ready to receive data.
AO8C500K_IRQ_LOAD_READY_L2H	This refers to a circular buffer becoming ready to receive data.

4.7.35. AO8C500K_IOCTL_LOAD_READY

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

Usage

Argument	Description
request	AO8C500K_IOCTL_LOAD_READY
arg	s32*

Valid values returned by the service are as follows.

Value	Description
AO8C500K_LOAD_READY_NO	The buffer is not ready to receive additional data.
AO8C500K_LOAD_READY_YES	The buffer is ready to receive additional data.

4.7.36. AO8C500K_IOCTL_LOAD_REQUEST

This service requests that the buffer become ready to receive additional data when in circular buffer mode. The service waits for up to the write timeout period for the operation to complete (though not the infinite option). (See AO8C500K_IOCTL_TX_IO_TIMEOUT, section 4.7.49, page 42.)

Usage

Argument	Description
request	AO8C500K_IOCTL_LOAD_REQUEST
arg	Not used.

4.7.37. AO8C500K_IOCTL_OUTPUT_MODE

This service configures the buffer's data output mode.

Usage

Argument	Description
request	AO8C500K_IOCTL_OUTPUT_MODE
arg	s32*

Valid argument values are as follows.

Value	Description
-1	Retrieve the current setting.
AO8C500K_OUTPUT_MODE_IMMEDIATE	Channel data is output as soon as it is received.
AO8C500K_OUTPUT_MODE_SIMULTANEOUS	Channel data is output to all channels simultaneously.

4.7.38. AO8C500K_IOCTL_QUERY

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

Usage

Argument	Description
request	AO8C500K_IOCTL_QUERY
arg	s32*

Valid argument values are as follows.

Value	Description
AO8C500K_QUERY_AUTOCAL_MS	This returns the maximum duration of the Autocalibration cycle in milliseconds.
AO8C500K_QUERY_CHANNEL_MAX	This returns the maximum number of output channels supported by the board.
AO8C500K_QUERY_CHANNEL_QTY	This returns the actual number of output channels on the current board. This will be either four or eight.
AO8C500K_QUERY_COUNT	This returns the number of query options supported by the IOCTL service.
AO8C500K_QUERY_DEVICE_TYPE	This returns the identifier value for the board's type. The value should be GSC_DEV_TYPE_20AO8C500K.
AO8C500K_QUERY_FGEN_MAX	This gives the maximum rate generator output frequency.
AO8C500K_QUERY_FGEN_MIN	This gives the minimum rate generator output frequency.
AO8C500K_QUERY_FIFO_SIZE	This returns the size of the output buffer in samples.
AO8C500K_QUERY_FREF_DEFAULT	This gives the default FREF value in hertz.
AO8C500K_QUERY_FSAMP_MAX	This gives the maximum supported sample rate in S/S.
AO8C500K_QUERY_FSAMP_MIN	This gives the minimum supported sample rate in S/S.
AO8C500K_QUERY_INIT_MS	This returns the duration of a board initialization in milliseconds.
AO8C500K_QUERY_NDIV_MAX	This returns the maximum supported rate generator NDIV value.
AO8C500K_QUERY_NDIV_MIN	This returns the minimum supported rate generator NDIV value.
AO8C500K_QUERY_OUTPUT_DISCONNECT	This indicates if the Output Disconnect feature is supported. Zero means it isn't. Non-zero means it is.
AO8C500K_QUERY_OUTPUT_FILTER	This gives the output filter cutoff frequency in hertz.
AO8C500K_QUERY_OUTPUT_RANGE	This indicates the voltage range supported by the board. See below.
AO8C500K_QUERY_PIN_A33	This indicates the functionality supported by cable interface pin A33. See below.

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

Value	Description
AO8C500K_IOCTL_QUERY_ERROR	Either there was a processing error or the query option is unrecognized.

Values returned for the AO8C500K_QUERY_OUTPUT_RANGE query option are as follows.

Value	Description
AO8C500K_QUERY_OUTPUT_RANGE_5	The board supports software selectable voltage ranges of ±5 volts and ±2.5 volts.
AO8C500K_QUERY_OUTPUT_RANGE_10	The board supports software selectable voltage ranges of ±10 volts and ±5 volts.

Values returned for the AO8C500K_QUERY_PIN_A33 query option are as follows.

Value	Description
AO8C500K_QUERY_PIN_A33_OUT_RTN	The pin is an Output Return input.
AO8C500K_QUERY_PIN_A33_RMT_GND_S	The pin is the Remote Ground Sense input.

4.7.39. AO8C500K_IOCTL_RANGE

This service configures the board's output voltage range.

Usage

Argument	Description
request	AO8C500K_IOCTL_RANGE
arg	s32*

Valid argument values are as follows.

Value	Description
-1	Retrieve the current setting.
AO8C500K_RANGE_2_5	This selects the ±2.5-volt range.
AO8C500K_RANGE_5	This selects the ±5-volt range.
AO8C500K_RANGE_10	This selects the ±10-volt range.

4.7.40. AO8C500K_IOCTL_REG_MOD

This service performs a read-modify-write of an 20AO8C500K register. This includes only the GSC firmware registers. The PCI and PLX Feature Set Registers are read-only. Refer to 20ao8c500k.h for a complete list of the GSC firmware registers.

Usage

Argument	Description
request	AO8C500K_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.

4.7.41. AO8C500K_IOCTL_REG_READ

This service reads the value of an 20AO8C500K register. This includes the PCI registers, the PLX Feature Set Registers and the GSC firmware registers. Refer to `20ao8c500k.h` and `gsc_pci9056.h` for the complete list of accessible registers.

Usage

Argument	Description
request	AO8C500K_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.

4.7.42. AO8C500K_IOCTL_REG_WRITE

This service writes a value to an 20AO8C500K register. This includes only the GSC firmware registers. The PCI and PLX Feature Set Registers are read-only. Refer to `20ao8c500k.h` for a complete list of the GSC firmware registers.

Usage

Argument	Description
request	AO8C500K_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.

4.7.43. AO8C500K_IOCTL_REMOTE_GND_SENSE

This service enables or disables the Remote Ground Sense feature if the A33 connector pin functions as a Remote Ground Sense input. (The function of the A33 connector pin is a board ordering option.)

Usage

Argument	Description
request	AO8C500K_IOCTL_REMOTE_GND_SENSE
arg	s32*

Valid argument values are as follows.

Value	Description
-1	Retrieve the current setting. †
AO8C500K_REMOTE_GND_SENSE_NO	This disables the Remote Ground Sense feature. †
AO8C500K_REMOTE_GND_SENSE_YES	This enables the Remote Ground Sense feature.

† The value -1 is returned if the *yes* option is requested when the Remote Ground Sense feature is unsupported.

4.7.44. AO8C500K_IOCTL_TRIGGER_SW

This service initiates an output burst cycle. The service waits for up to the write timeout period for the operation to complete (though not the infinite option). (See AO8C500K_IOCTL_TX_IO_TIMEOUT, section 4.7.49, page 42.)

Usage

Argument	Description
request	AO8C500K_IOCTL_TRIGGER_SW
arg	Not used.

4.7.45. AO8C500K_IOCTL_TX_IO_ABORT

This service aborts an ongoing write() request.

Usage

Argument	Description
request	AO8C500K_IOCTL_TX_IO_ABORT
arg	s32*

The results are reported as one of the following values.

Value	Description
AO8C500K_IO_ABORT_NO	A `write()` request was not aborted as none were ongoing.
AO8C500K_IO_ABORT_YES	An ongoing `write()` request was aborted.

4.7.46. AO8C500K_IOCTL_TX_IO_MODE

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

> **NOTE:** Applications may experience improved responsiveness with write requests by coordinating the buffer Threshold setting with the number of samples in the write request.

Usage

Argument	Description
request	AO8C500K_IOCTL_TX_IO_MODE
arg	s32*

Valid argument values are as follows.

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

4.7.47. AO8C500K_IOCTL_TX_IO_OVER_DATA

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

Usage

Argument	Description
request	AO8C500K_IOCTL_TX_IO_OVER_DATA
arg	s32*

Valid argument values are as follows.

Value	Description
-1	Retrieve the current setting.
AO8C500K_TX_IO_OVER_CHECK	Perform the check. This is the default.
AO8C500K_TX_IO_OVER_IGNORE	Do not perform the check.

4.7.48. AO8C500K_IOCTL_TX_IO_OVER_FRAME

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

Usage

Argument	Description
request	AO8C500K_IOCTL_TX_IO_OVER_FRAME
arg	s32*

Valid argument values are as follows.

Value	Description
-1	Retrieve the current setting.
AO8C500K_TX_IO_OVERFLOW_CHECK	Perform the check. This is the default.
AO8C500K_TX_IO_OVERFLOW_IGNORE	Do not perform the check.

4.7.49. AO8C500K_IOCTL_TX_IO_TIMEOUT

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

Usage

Argument	Description
request	AO8C500K_IOCTL_TX_IO_TIMEOUT
arg	s32*

Valid argument values are in the range from zero to 3600, -1, and AO8C500K_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 AO8C500K_IO_TIMEOUT_INFINITE is used, then the driver will wait indefinitely rather than timing out. The default is 10 seconds.

4.7.50. AO8C500K_IOCTL_WAIT_CANCEL

This service resumes all threads blocked via AO8C500K_IOCTL_WAIT_EVENT IOCTL calls (section 4.7.51, page 43), 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	AO8C500K_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 4.7.51.2 on page 44.
gsc	This specifies the set of AO8C500K_WAIT_GSC_* events whose wait requests are to be cancelled. Refer to section 4.7.51.3 on page 44.

alt	This is unused by the 20AO8C500K driver and should be zero.
io	This specifies the set of AO8C500K_WAIT_IO_* events whose wait requests are to be cancelled. Refer to section 4.7.51.4 on page 45.
timeout_ms	This is unused by wait cancel operations.
count	Upon return this indicates the number of waits that were cancelled.

4.7.51. AO8C500K_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	AO8C500K_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 4.7.51.1on page 44.
main	This specifies any number of GSC_WAIT_MAIN_* events that the thread is to wait for. Refer to section 4.7.51.2 on page 44.
gsc	This specifies any number of AO8C500K_WAIT_GSC_* events that the thread is to wait for. Refer to section 4.7.51.3 on page 44.
alt	This is unused by the 20AO8C500K driver and must be zero.
io	This specifies any number of AO8C500K_WAIT_IO_* events that the thread is to wait for. Refer to section 4.7.51.4 on page 45.
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.

4.7.51.1. 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.

4.7.51.2. 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 20AO8C500K 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 20AO8C500K.
GSC_WAIT_MAIN_PCI	This refers to any interrupt generated by the 20AO8C500K.
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.

4.7.51.3. gsc_wait_t.gsc Options

The wait structure's gsc field may specify any combination of the below interrupt options. These are the interrupt options referenced in the Board Control Register. Applications are responsible for selecting the desired interrupt options. Refer to AO8C500K_IOCTL_IRQ_ENABLE (section 4.7.34, page 35).

Value	Description
AO8C500K_WAIT_GSC_AUTOCAL_DONE	This refers to the completion of autocalibration.
AO8C500K_WAIT_GSC_BUF_ERROR	This refers to the occurrence of a buffer overflow or a frame overflow.
AO8C500K_WAIT_GSC_BUF_THR_H2L	This refers to buffer fill level falling to the Threshold level.
AO8C500K_WAIT_GSC_BUF_THR_L2H	This refers to buffer fill level rising above the Threshold level.
AO8C500K_WAIT_GSC_BURST_READY	This refers to the board becoming ready for a burst trigger.

AO8C500K_WAIT_GSC_DIO0_L2H	This refers to a low to high transition on digital I/O line D0.
AO8C500K_WAIT_GSC_LD_RDY_H2L	This refers to a circular buffer becoming not ready to receive data.
AO8C500K_WAIT_GSC_LD_RDY_L2H	This refers to a circular buffer becoming ready to receive data.

4.7.51.4. 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
AO8C500K_WAIT_IO_TX_ABORT	This refers to write requests which have been aborted.
AO8C500K_WAIT_IO_TX_DONE	This refers to write requests which have been satisfied.
AO8C500K_WAIT_IO_TX_ERROR	This refers to write requests which end due to an error.
AO8C500K_WAIT_IO_TX_TIMEOUT	This refers to write requests which end due to the timeout period lapse.

4.7.52. AO8C500K_IOCTL_WAIT_STATUS

This service counts all threads blocked via the AO8C500K_IOCTL_WAIT_EVENT IOCTL service (section 4.7.51, page 43), 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	AO8C500K_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 4.7.51.2 on page 44.
gsc	This specifies the set of AO8C500K_WAIT_GSC_* events whose wait requests are to be counted. Refer to section 4.7.51.3 on page 44.
alt	This is unused by the 20AO8C500K driver and should be zero.
io	This specifies the set of AO8C500K_WAIT_IO_* events whose wait requests are to be counted. Refer to section 4.7.51.4 on page 45.
timeout_ms	This is unused by wait status operations.
count	Upon return this indicates the number of waits that met any of the specified criteria.

4.7.53. AO8C500K_IOCTL_XCVR_OUTPUT

This service enables or disables the clock and trigger output signals at the cable interface.

Usage

Argument	Description
request	AO8C500K_IOCTL_XCVR_OUTPUT
arg	s32*

Valid argument values are as follows.

Value	Description
-1	Retrieve the current setting. †
AO8C500K_XCVR_OUTPUT_DISABLE	This disables the transceivers. †
AO8C500K_XCVR_OUTPUT_ENABLE	This enables the transceivers.

† The value -1 is returned if the *disable* option is requested when the Output Disconnect feature is unsupported.

4.7.54. AO8C500K_IOCTL_XCVR_POL

This service configures the polarity of the clock and trigger signals at the cable interface. This affects the clock and trigger signals on the System I/O Connector, but not those on the Auxiliary SYNC I/O Connector.

Usage

Argument	Description
request	AO8C500K_IOCTL_XCVR_POL
arg	s32*

Valid argument values are as follows.

Value	Description
-1	Retrieve the current setting.
AO8C500K_XCVR_POL_DEFAULT	This selects the signals' default polarities.
AO8C500K_XCVR_POL_INVERT	This inverts the signals from their default polarities.

4.7.55. AO8C500K_IOCTL_XCVR_TYPE

This service selects TTL or LVDS signaling on the external clock and trigger signals. This affects the clock and trigger signals on the System I/O Connector as well as those on the Auxiliary SYNC I/O Connector.

Usage

Argument	Description
request	AO8C500K_IOCTL_XCVR_TYPE
arg	s32*

Valid argument values are as follows.

Value	Description
-1	Retrieve the current setting.
AO8C500K_XCVR_TYPE_LVDS	Use LVDS signaling.
AO8C500K_XCVR_TYPE_TTL	Use TTL signaling.

5. The Driver

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

5.1. Files

The device driver files are summarized in the table below.

Description	Files	Location
Source Files	`*.c`, `*.h` …	…`/driver/`
Header File	`20ao8c500k.h`	
Driver File	`20ao8c500k.ko` † `20ao8c500k.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 20AO8C500K 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 `20ao8c500k` 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/20ao8c500k.*
   ```

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/20ao8c500k/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/20ao8c500k` 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/20ao8c500k
   ```

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/20ao8c500k` 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 20ao8c500k
   ```

2. Verify that the driver module has been unloaded by issuing the below command. The module name `20ao8c500k` 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	`20ao8c500k_dsl.h`	`…/include/`
Library File	`20ao8c500k_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 `ao8c500k_open()` there is the utility file `open.c` containing the utility function `ao8c500k_open_util()`. The naming pattern is as follows: API function `ao8c500k_xxxx()`, utility file name `xxxx.c`, utility function `ao8c500k_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 `AO8C500K_IOCTL_QUERY` there is the utility file `query.c` containing the utility function `ao8c500k_query()`. The naming pattern is as follows: IOCTL code `AO8C500K_IOCTL_XXXX`, utility file name `xxxx.c`, utility function `ao8c500k_xxxx()`.

7.1. Files

The utility files are summarized in the table below.

Description	Files	Location
Source Files	`*.c, *.h …`	`…/utils/`
Header File	`20ao8c500k_utils.h`	`…/include/`

Library Files	`20ao8c500k_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 20AO8C500K. 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	`ao8c500k_reg_list()`	Source File
Source File	`util_reg.c`	`…/utils/`
Header File	`20ao8c500k_utils.h`	`…/include/`
Library File	`20ao8c500k_utils.a`	`…/lib/`

8.2. Analog Output Configuration

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

Item	Name/File	Location
Function	`ao8c500k_config_ao()`	Source File
Source File	`util_config_ai.c`	`…/utils/`
Header File	`20ao8c500k_utils.h`	`…/include/`
Library File	`20ao8c500k_utils.a`	`…/lib/`

8.3. Auxiliary I/O Configuration

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

Item	Name/File	Location
Function	ao8c500k_config_aux()	Source File
Source File	util_config_ai.c	…/utils/
Header File	20ao8c500k_utils.h	…/include/
Library File	20ao8c500k_utils.a	…/lib/

8.4. Digital I/O Configuration

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

Item	Name/File	Location
Function	ao8c500k_config_dio()	Source File
Source File	util_config_ai.c	…/utils/
Header File	20ao8c500k_utils.h	…/include/
Library File	20ao8c500k_utils.a	…/lib/

8.5. Data Transfer Modes

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

8.5.1. PIO

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

8.5.2. Block Mode DMA

For Block Mode DMA transfers, hardware onboard the 20AO8C500K is used to transfer the data without processor intervention. In this mode the driver checks for available space in the output buffer. Depending on the size of the write request, the driver may break the request into smaller transfers in order to ensure data integrity. When sufficient space is available a DMA transfer is performed. The volume of data moved in a single request is based upon the amount of data remaining in the request and the amount of space available in the buffer. If the remaining request will fit within the available space, then the data is transferred. Otherwise, the driver will wait one timer tick before trying again. The process is repeated until the data is exhausted or the I/O timeout expires, whichever occurs first.

8.5.3. 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 space becomes available in the output buffer. 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 located under the `samples` subdirectory. While they are provided without support and without any external documentation, any problems reported will be addressed as time permits. The applications are command line based and produce text output for display on a console. All of the applications are built via the Overall Make Script (section 2.7, page 13), but each may be built individually by changing to its respective directory and issuing the commands "`make clean`" and "`make`". The initial output from each application includes information on its supported command line arguments. The following gives a brief overview of each application.

9.1. aout - Analog Output - …/aout/

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

9.2. din - Digital Input - …/din/

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

9.3. dout - Digital Output - …/dout/

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

9.4. 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. irq - Interrupt Test - …/irq/

This application performs complete testing to verify the operation of the board's firmware interrupts.

9.7. regs - Register Access - …/regs/

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

9.8. signals - Digital Signals - …/signals/

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

9.9. txrate – Transmit Rate - …/txrate/

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

Document History

Revision	Description
September 21, 2023	Initial release, version 1.0.105.47.0.