24DSI6C500K

24-bit, 6 Channel, 500KS/S/Ch Delta-Sigma A/D Input

PMC66-24DSI6C500K

Linux Device Driver
And API Library
User Manual

Manual Revision: July 8, 2024
Driver Release Version 2.4.111.50.0

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

Preface

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

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

Table of Contents

Table of Figures

1. Introduction

1.1. Purpose

The purpose of this document is to describe the interface to the 24DSI6C500K 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 24DSI6C500K 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
DMA	Direct Memory Access
DMDMA	Demand Mode DMA
GSC	General Standards Corporation
PCI	Peripheral Component Interconnect
PIO	Programmed I/O
PMC	PCI Mezzanine Card
PMC66	This is a PMC formfactor device that can operate at up to 66MHz over the PCI bus.

1.3. Definitions

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

Term	Definition
…	This is a shortcut representation of the 24DSI6C500K installation directory or any of its subdirectories.
24DSI6C500K	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 24DSI6C500K hardware.
Application	This is a user mode process, which runs in user space with user mode privileges.
Driver	This is the 24DSI6C500K 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 24DSI6C500K applications. The overall architecture is illustrated in Figure 1 below.

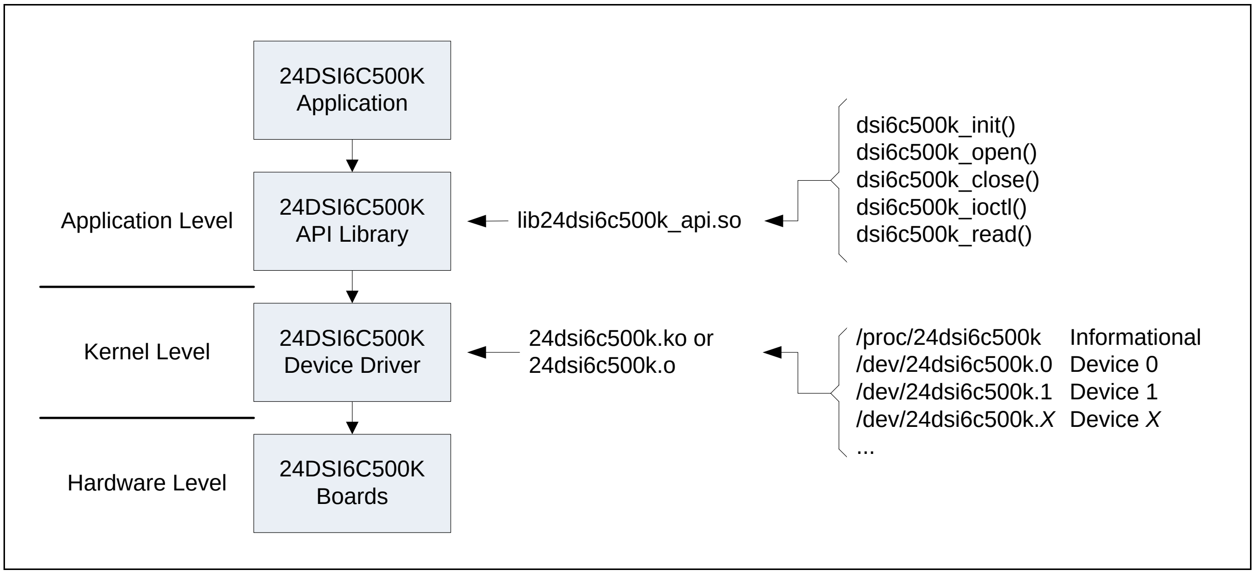

Figure 1 Basic architectural representation.

1.4.2. API Library

The primary means of accessing 24DSI6C500K boards is via the 24DSI6C500K API Library. This library forms a layer between the application and the driver. Additional information is given in section 3.2.3 (page 17). With the library, applications are able to open and close a device and, while open, perform I/O control and read operations.

1.4.3. Device Driver

The device driver is the host software that provides a means of communicating directly with 24DSI6C500K 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 24DSI6C500K is a high-performance, 24-bit analog input board that contains six input channels. The host side connection is PCI based and the form factor is according to the model ordered. The board is capable of acquiring data at up to 500K samples per second over each channel. Internal clocking permits sampling rates from 500K samples per second down to just under 10K samples per second. Onboard storage permits data buffering of up to 256K samples, for all channels collectively, between the cable interface and the PCI bus. This allows the 24DSI6C500K to sustain continuous throughput from the cable interface independent of the PCI bus interface. The 24DSI6C500K also permits multiple boards to be synchronized so that all boards sample data in unison. In addition, the board includes autocalibration capability and eight general purpose digital I/O lines.

1.6. Reference Material

The following reference material may be of particular benefit in using the 24DSI6C500K. The specifications provide the information necessary for an in depth understanding of the specialized features implemented on this board.

- The applicable *24DSI6C500K User Manual* from General Standards Corporation.
- The *PCI9056 PCI Bus Master Interface Chip* data handbook from PLX Technology, Inc.

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

1.7. Licensing

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

2. Installation

2.1. CPU and Kernel Support

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

Kernel	Distribution
6.2.9	Red Hat Fedora Core 38
6.0.7	Red Hat Fedora Core 37
5.17.5	Red Hat Fedora Core 36
5.14.10	Red Hat Fedora Core 35
5.11.12	Red Hat Fedora Core 34
5.8.15	Red Hat Fedora Core 33
5.6.6	Red Hat Fedora Core 32
5.3.7	Red Hat Fedora Core 31
5.0.9	Red Hat Fedora Core 30
4.18.16	Red Hat Fedora Core 29
4.16.3	Red Hat Fedora Core 28
4.13.9	Red Hat Fedora Core 27
4.11.8	Red Hat Fedora Core 26
4.8.6	Red Hat Fedora Core 25
4.5.5	Red Hat Fedora Core 24
4.2.3	Red Hat Fedora Core 23
4.0.4	Red Hat Fedora Core 22
3.17.4	Red Hat Fedora Core 21
3.11.10	Red Hat Fedora Core 20
3.9.5	Red Hat Fedora Core 19
3.6.10	Red Hat Fedora Core 18
3.3.4	Red Hat Fedora Core 17
3.1.0	Red Hat Fedora Core 16
2.6.38	Red Hat Fedora Core 15
2.6.35	Red Hat Fedora Core 14
2.6.33	Red Hat Fedora Core 13
2.6.31	Red Hat Fedora Core 12
2.6.29	Red Hat Fedora Core 11
2.6.27	Red Hat Fedora Core 10
2.6.25	Red Hat Fedora Core 9
2.6.23	Red Hat Fedora Core 8
2.6.21	Red Hat Fedora Core 7
2.6.18	Red Hat Fedora Core 6
2.6.15	Red Hat Fedora Core 5
2.6.11	Red Hat Fedora Core 4
2.6.9	Red Hat Fedora Core 3
2.4.18	Red Hat 8.0

> **NOTE:** Some older kernel versions are supported (the sources are maintained), but are not tested.
>
> **NOTE:** While only Red Hat Fedora distributions are listed, numerous other distributions are supported and have been tested on an as needed basis.
>
> **NOTE:** The driver will have to be built before being used as it is provided in source form only.

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

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

2.1.1. 32-bit Support Under 64-bit Environments

This driver supports 32-bit applications under 64-bit environments. The availability of this feature in the kernel depends on a 64-bit kernel being configured to support 32-bit application compatibility. Additionally, 2.6 kernels prior to 2.6.11 implemented 32-bit compatibility in a way that resulted in some drivers not being able to take advantage of the feature. (In these kernels a driver's IOCTL command codes must be globally unique. Beginning with 2.6.11 this requirement has been lifted.) If the driver is not able to provide 32-bit support under a 64-bit kernel, the "`32-bit support`" field in the `/proc/24dsi6c500k` file will be "`no`".

2.2. The /proc/ File System

While the driver is running, the text file `/proc/24dsi6c500k` 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: 2.4.111.50
32-bit support: yes
boards: 1
models: 24DSI6C500K
```

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
`24dsi6c500k.linux.tar.gz`	This archive contains the driver, the API Library and all related files.
`24dsi6c500k_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
`24dsi6c500k/`	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 3.2.3, page 17).
…`/docsrc/`	This directory contains the source files for the code samples given in this document (section 6, page 48).
…`/driver/`	This directory contains the device driver source files (section 5, page 44).
…`/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 53).
`…/utils/`	This directory contains the source files for the utility libraries used by the sample applications (section 7, page 49).

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 `24dsi6c500k.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 `24dsi6c500k` in the current directory, and then copies all of the archive's files into this new directory.

```
tar –xzvf 24dsi6c500k.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 47).

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 24dsi6c500k.linux.tar.gz 24dsi6c500k
```

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

```
rm -f /dev/24dsi6c500k.*
```

5. If the automatic startup procedure was adopted (section 5.3.2, page 45), then edit the system startup script `rc.local` and remove the line that invokes the 24DSI6C500K'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 (…`/24dsi6c500k/`).
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/lib24dsi6c500k_api.so`
Defined and Not Empty	`==== Linking: ../lib/lib24dsi6c500k_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/24dsi6c500k_utils.a`
Defined and Not Empty	`==== Linking: ../lib/24dsi6c500k_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 24DSI6C500K 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 24DSI6C500K 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 24DSI6C500K specific header files. Therefore, sources may include only this one 24DSI6C500K header and make files may reference only this one 24DSI6C500K include directory.

Description	File	Location
Header File	`24dsi6c500k_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 24DSI6C500K 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 24DSI6C500K static library and only this one 24DSI6C500K library directory.

Description	File	Location
Static Library	`24dsi6c500k_main.a`	`…/lib/`
	`24dsi6c500k_multi.a`	

> **NOTE**: For applications using the 24DSI6C500K and no other GSC devices, link the `24dsi6c500k_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 24DSI6C500K API Library is implemented as a shared library and is thus not linked with the 24DSI6C500K Main Library. The API Library must be linked with applications by adding the argument `-l24dsi6c500k_api` to the linker command line.

3.2.1. Build

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

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

2. Remove existing build targets using the below command.

   ```
   make clean
   ```

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

   ```
   make
   ```

3.2.2. System Libraries

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

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

3.2.3. Shared Object Script: Build the Main Libraries as Shard Object Files

The main libraries built via the Overall Make Script (section 2.7, page 13) are static library files. Some applications however, require that the Main Libraries be accessed as shared object files. Generating shared object files require that all of the static libraries be recompiled for this purpose and linked as .so files. This is done using the Shared Object Script named below. When run, the script invokes the Overall Make Script to clean all existing build targets, deletes the two shared object files named below, if they exist, defines an environment variable used by all of the static library make files, invokes the Overall Make Script again to rebuild all existing build targets then invokes make on the library make file (…`/lib/makefile`) to link the shared object files. The required manual steps are as follows.

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

2. Execute the below script.

```
./static_to_shared.sh
```

Running the above-named Shared Object Script produces the files given in the table below. These shared object files fulfill the same purpose as the similarly named static libraries as described in the note under section 3.2 above. Refer to that note when selecting which shared object file to use.

Description	File	Location

Shared Object Files	`lib16ai32sc_main.so`
	`lib16ai32sc_multi.so`
	`lib16ai32sc_all.so` †
		…`/lib/`

† This library includes all generated libraries, including the API Library shared object file content.

The shared object files can be linked via two different methods. In the first method, the application linker command line can explicitly name the file in the same manner as is done were it a static library. This is the method used by the sample applications, all of which use the SIO4 API Library, which itself is a shared object file. This file is also found in the …`/lib/` subdirectory. In the second method, the .so files are copied to the `/usr/lib/` subdirectory and are referenced on the application's liker command line as given in the table below.

Library	gcc Link Flag
`lib16ai32sc_main.so`	`-lsio4_main`
`lib16ai32sc_multi.so`	`-lsio4_multi`
`lib16ai32sc_all.so` †	`-lsio4_all`

† This library includes all generated libraries, including the API Library shared object file content.

4. API Library

The 24DSI6C500K API Library is the software interface between user applications and the 24DSI6C500K device driver. The interface is accessed by including the header file `24dsi6c500k_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	`24dsi6c500k_api.h`	…`/include/`
Library File	`lib24dsi6c500k_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	`24dsi6c500k_api.h`	…`/include/`	
Shared Object Library	`lib24dsi6c500k_api.so`	…`/lib/`	
		`/usr/lib/`	`-l24dsi6c500k_api`

4.4. Macros

The API Library and driver interfaces include the following macros, which are defined in `24dsi6c500k.h`.

4.4.1. IOCTL Services

The IOCTL macros are documented in section 4.7 (page 24).

4.4.2. Registers

The following gives the complete set of 24DSI6C500K registers.

4.4.2.1. GSC Registers

The following table gives the complete set of GSC specific 24DSI6C500K 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 *24DSI6C500K 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
`DSI6C500K_GSC_AVR`	Autocal Values Register (AVR)
`DSI6C500K_GSC_BBSR`	Burst Block Size Register (BBSR)
`DSI6C500K_GSC_BCFGR`	Board Configuration Register (BCFGR)
`DSI6C500K_GSC_BCTLR`	Board Control Register (BCTLR)
`DSI6C500K_GSC_BTTR`	Burst Trigger Timer Register (BTTR)
`DSI6C500K_GSC_BUFCR`	Buffer Control Register (BUFCR)
`DSI6C500K_GSC_BUFSR`	Buffer Size Register (BUFSR)
`DSI6C500K_GSC_CSAR`	Clock Source Assignment Register (CSAR)
`DSI6C500K_GSC_DIOPR`	Digital I/O Port Register (DIOPR)
`DSI6C500K_GSC_IDBR`	Input Data Buffer Register (IDBR)
`DSI6C500K_GSC_RCR`	Rate Control Register (RCR)
`DSI6C500K_GSC_RDR`	Rate Divisors Register (RDR)

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 `24dsi6c500k_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 `24dsi6c500k_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 24).

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. dsi6c500k_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 dsi6c500k_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 "24dsi6c500k_dsl.h"

int dsi6c500k_close_dsl(int fd)
{
    int errs;
    int ret;

    ret = dsi6c500k_close(fd);

    if (ret)
        printf("ERROR: dsi6c500k_close() returned %d\n", ret);

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

4.6.2. dsi6c500k_init()

This function is the entry point to initializing the 24DSI6C500K 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 dsi6c500k_init(void);
```

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

Example

```
#include <stdio.h>

#include "24dsi6c500k_dsl.h"

int dsi6c500k_init_dsl(void)
{
    int errs;
    int ret;

    ret = dsi6c500k_init();

    if (ret)
        printf("ERROR: dsi6c500k_init() returned %d\n", ret);

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

4.6.3. dsi6c500k_ioctl()

This function is the entry point to performing setup and control operations on a 24DSI6C500K. 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 24).

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

NOTE: Some of the driver's IOCTL services wait for the board's Ready Bit in the Board Control Register to become set after applying the requested settings. If the respective board feature requires a clock source and the clock source is absent or disabled, then the service may fail with a timeout error. This is most likely to occur if the required clock source is disabled or if the external source is not providing a clock.

Prototype

```
int dsi6c500k_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 24).
`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 "24dsi6c500k_dsl.h"

int dsi6c500k_ioctl_dsl(int fd, int request, void* arg)
{
    int errs;
    int ret;

    ret = dsi6c500k_ioctl(fd, request, arg);

    if (ret)
        printf("ERROR: dsi6c500k_ioctl() returned %d\n", ret);

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

4.6.4. dsi6c500k_open()

This function is the entry point to open a connection to a 24DSI6C500K board. Before returning, the initialize IOCTL service is called to reset all hardware and software settings to their defaults.

Prototype

```
int dsi6c500k_open(int device, int share, int* fd);
```

Argument	Description
`device`	This is the zero-based index of the 24DSI6C500K to access. †
`share`	Open the device in Shared Access Mode? If non-zero the device is opened in Shared Access Mode (see below). If zero the device is opened in Exclusive Access Mode (see below).
`fd`	The device handle is returned here. The pointer cannot be NULL. Values returned are as follows. <table><tr><th>Value</th><th>Description</th></tr><tr><td>>= 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>

† 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 "24dsi6c500k_dsl.h"

int dsi6c500k_open_dsl(int device, int share, int* fd)
{
    int errs;
    int ret;

    ret = dsi6c500k_open(device, share, fd);

    if (ret)
        printf("ERROR: dsi6c500k_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. dsi6c500k_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 8.3, page 50).
>
> **NOTE:** Applications may experience improved responsiveness with read requests by coordinating the Buffer Threshold with the number of samples requested. Refer to the `DSI6C500K_IOCTL_IN_BUF_THRESH` service of section 4.7.5 on page 25. Refer also to the Data Reception portion of the operations section (section 8.1, page 50).

Prototype

```
int dsi6c500k_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 4.7.39, page 38) before the entire request could be satisfied.
`< 0`	An error occurred. See error value description above.

Example

```
#include <stdio.h>

#include "24dsi6c500k_dsl.h"

int dsi6c500k_read_dsl(int fd, void* dst, size_t bytes, size_t* qty)
{
    int errs;
    int ret;

    ret = dsi6c500k_read(fd, dst, bytes);

    if (ret < 0)
        printf("ERROR: dsi6c500k_read() 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 24DSI6C500K API Library and device driver implement the following IOCTL services. Each service is described along with the applicable `dsi6c500k_ioctl()` function arguments.

4.7.1. DSI6C500K_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`	`DSI6C500K_IOCTL_AI_BUF_CLEAR`
`arg`	Not used.

4.7.2. DSI6C500K_IOCTL_AI_BUF_ENABLE

This service enables or disables input to the analog input buffer.

Usage

Argument	**Description**
`request`	`DSI6C500K_IOCTL_AI_BUF_ENABLE`
`arg`	`s32*`

Valid argument values are as follows.

Value	Description
`-1`	Retrieve the current setting.
`DSI6C500K_AI_BUF_ENABLE_NO`	This option disables input to the input buffer.
`DSI6C500K_AI_BUF_ENABLE_YES`	This option enables input to the input buffer.

4.7.3. DSI6C500K_IOCTL_AI_BUF_FILL_LVL

This service reports the analog input buffer's current fill level.

Usage

Argument	Description
`request`	`DSI6C500K_IOCTL_AI_BUF_FILL_LVL`
`arg`	`s32*`

Valid return values are from zero to `0x80000` (512K).

4.7.4. DSI6C500K_IOCTL_AI_BUF_OVERFLOW

This service operates on the Analog Input Overflow status.

Usage

Argument	Description
`request`	`DSI6C500K_IOCTL_AI_BUF_OVERFLOW`
`arg`	`s32*`

Valid argument values provided to the service are as follows.

Value	Description
`DSI6C500K_AI_BUF_OVERFLOW_CLEAR`	This option clears the overflow status.
`DSI6C500K_AI_BUF_OVERFLOW_TEST`	This option reports if an overflow has occurred.

Valid returned values are as follows.

Value	Description
`DSI6C500K_AI_BUF_OVERFLOW_NO`	An overflow did not occur.
`DSI6C500K_AI_BUF_OVERFLOW_YES`	An overflow did occur.

4.7.5. DSI6C500K_IOCTL_AI_BUF_THRESH

This service sets the receive buffer threshold level for read operations. When DMA read requests necessitate waiting for data, the read typically waits for the input buffer to fill to this level before initiating the DMA transfer.

> **NOTE:** Applications may experience improved responsiveness with read requests if the number of samples requested equals the Buffer Threshold level.

Usage

Argument	Description
`request`	`DSI6C500K_IOCTL_AI_BUF_THRESH`
`arg`	`s32*`

Valid argument values are from zero to `0x80000`, and -1. A value of -1 will return the current threshold level setting.

4.7.6. DSI6C500K_IOCTL_AI_BUF_THR_STS

This service reports the input buffer threshold status.

Usage

Argument	Description
`request`	`DSI6C500K_IOCTL_AI_BUF_THR_STS`
`arg`	`s32*`

Valid returned values are as follows.

Value	Description
`DSI6C500K_AI_BUF_THR_STS_ACTIVE`	The threshold flag is set.
`DSI6C500K_AI_BUF_THR_STS_IDLE`	The threshold flag is not set.

4.7.7. DSI6C500K_IOCTL_AI_BUF_UNDERFLOW

This service operates on the Analog Input Underflow status.

Usage

Argument	Description
`request`	`DSI6C500K_IOCTL_AI_BUF_UNDERFLOW`
`arg`	`s32*`

Valid argument values are as follows.

Value	Description
`DSI6C500K_AI_BUF_UNDERFLOW_CLEAR`	Clear the underflow status.
`DSI6C500K_AI_BUF_UNDERFLOW_TEST`	Report if an underflow has occurred.

Valid returned values are as follows.

Value	Description
`DSI6C500K_AI_BUF_UNDERFLOW_NO`	An underflow did not occur.
`DSI6C500K_AI_BUF_UNDERFLOW_YES`	An underflow did occur.

4.7.8. DSI6C500K_IOCTL_AI_CHANNEL_TAG

This service configures the appearance of the channel tag in the input data stream.

Usage

Argument	Description
`request`	`DSI6C500K_IOCTL_AI_CHANNEL_TAG`
`arg`	`s32*`

Valid argument values are as follows.

Value	Description
`-1`	Request the current setting.
`DSI6C500K_AI_CHANNEL_TAG_DISABLE`	This option causes the channel tag to not appear.
`DSI6C500K_AI_CHANNEL_TAG_ENABLE`	This option causes the channel tag to appear.

4.7.9. DSI6C500K_IOCTL_AI_MODE

This service configures the analog input mode.

Usage

Argument	Description
`request`	`DSI6C500K_IOCTL_AI_MODE`
`arg`	`s32*`

Valid argument values are as follows.

Value	Description
`-1`	Retrieve the current setting.
`DSI6C500K_AI_MODE_DIFF`	This option selects differential operation.
`DSI6C500K_AI_MODE_VREF`	This option connects the input channels to the onboard VREF signal.
`DSI6C500K_AI_MODE_ZERO`	This option connects the input channels to the onboard zero voltage signal.

4.7.10. DSI6C500K_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`	`DSI6C500K_IOCTL_AUTOCAL`
`arg`	Not used.

4.7.11. DSI6C500K_IOCTL_AUTOCAL_STS

This service reports the autocalibration status.

Usage

Argument	Description
`request`	`DSI6C500K_IOCTL_AUTOCAL_STS`
`arg`	`s32*`

Valid argument values returned are as follows.

Value	**Description**
`DSI6C500K_AUTOCAL_STS_ACTIVE`	An autocalibration cycle is in progress.
`DSI6C500K_AUTOCAL_STS_FAIL`	The most recent autocalibration cycle failed.
`DSI6C500K_AUTOCAL_STS_PASS`	The most recent autocalibration cycle passed.

4.7.12. DSI6C500K_IOCTL_BURST

This service enables or disables input bursting.

Usage

Argument	**Description**
`request`	`DSI6C500K_IOCTL_BURST`
`arg`	`s32*`

Valid argument values are as follows.

Value	**Description**
`-1`	Retrieve the current setting.
`DSI6C500K_BURST_DISABLE`	This option disables input bursting.
`DSI6C500K_BURST_ENABLE`	This option enables input bursting. †

† When bursting is enabled, the SYNC input functions as the Burst Trigger.

4.7.13. DSI6C500K_IOCTL_BURST_RATE_DIV

This service adjusts the Burst Rate Divisor.

Usage

Argument	**Description**
`request`	`DSI6C500K_IOCTL_BURST_RATE_DIV`
`arg`	`s32*`

Valid argument values are from zero to `0xFFFFFF`, and `-1`. The value `-1` retrieves the current setting.

4.7.14. DSI6C500K_IOCTL_BURST_SIZE

This service adjusts the Burst Size, which is the number of scans in a single burst operation.

Usage

Argument	**Description**
`request`	`DSI6C500K_IOCTL_BURST_SIZE`
`arg`	`s32*`

Valid argument values are from zero to `0xFFFFFF`, and `-1`. The value `-1` retrieves the current setting.

4.7.15. DSI6C500K_IOCTL_BURST_TIMER

This service enables or disables the input bursting timer.

Usage

Argument	Description
`request`	`DSI6C500K_IOCTL_BURST_TIMER`
`arg`	`s32*`

Valid argument values are as follows.

Value	Description
`-1`	Retrieve the current setting.
`DSI6C500K_BURST_TIMER_DISABLE`	This option disables the input burst timer.
`DSI6C500K_BURST_TIMER_ENABLE`	This option enables the input burst timer.

4.7.16. DSI6C500K_IOCTL_BURST_TRIGGER

This service initiates a burst operation.

Usage

Argument	Description
`request`	`DSI6C500K_IOCTL_BURST_TRIGGER`
`arg`	Not used.

4.7.17. DSI6C500K_IOCTL_CHANNELS_READY

This service operates on the Channel Ready status.

Usage

Argument	Description
`request`	`DSI6C500K_IOCTL_CHANNELS_READY`
`arg`	`s32*`

Valid argument values are as follows.

Value	Description
`DSI6C500K_CHANNELS_READY_TEST`	This reports if the status is *ready*.
`DSI6C500K_CHANNELS_READY_WAIT`	This requests that the driver wait for the status to become *ready*. The driver waits for up to one second.

Valid returned values are as follows.

Value	Description
`DSI6C500K_CHANNELS_READY_NO`	The status is *not ready*.
`DSI6C500K_CHANNELS_READY_YES`	The status is *ready*.

4.7.18. DSI6C500K_IOCTL_CH_GRP_*x*_SRC

This service configures the clocking source for the respective channel group.

NOTE: Refer to the board user manual for additional information.

Value	Description
`DSI6C500K_IOCTL_CH_GRP_0_SRC`	This refers to Channel Group 0.
`DSI6C500K_IOCTL_CH_GRP_1_SRC`	This refers to Channel Group 1.

Usage

Argument	Description
`request`	`DSI6C500K_IOCTL_CH_GRP_`*`X`*`_SRC`
`arg`	`s32*`

Valid argument values are as follows.

Value	Description
`-1`	Retrieve the current setting.
`DSI6C500K_CH_GRP_SRC_DIR_EXTERN`	This option selects the Direct External configuration.
`DSI6C500K_CH_GRP_SRC_DISABLE`	This option disables the respective channel group.
`DSI6C500K_CH_GRP_SRC_EXTERN`	This option selects the External configuration.
`DSI6C500K_CH_GRP_SRC_RATE_GEN`	This option selects the internal Rate Generator.

4.7.19. DSI6C500K_IOCTL_CONTROL_MODE

This service configures the board for initiator or target mode operation.

Usage

Argument	Description
`request`	`DSI6C500K_IOCTL_CONTROL_MODE`
`arg`	`s32*`

Valid argument values are as follows.

Value	Description
`-1`	Retrieve the current setting.
`DSI6C500K_CONTROL_MODE_INITIATOR`	This option selects initiator mode operation.
`DSI6C500K_CONTROL_MODE_TARGET`	This option selects target mode operation.

4.7.20. DSI6C500K_IOCTL_DATA_FORMAT

This service configures the data encoding format.

Usage

Argument	Description
`request`	`DSI6C500K_IOCTL_DATA_FORMAT`
`arg`	`s32*`

Valid argument values are as follows.

Value	Description
`-1`	Retrieve the current setting.
`DSI6C500K_DATA_FORMAT_2S_COMP`	This option selects the Twos Compliment data format.
`DSI6C500K_DATA_FORMAT_OFF_BIN`	This option selects the Offset Binary encoding format.

4.7.21. DSI6C500K_IOCTL_DATA_WIDTH

This service configures the bit width of the converted input data.

Usage

Argument	Description
`request`	`DSI6C500K_IOCTL_DATA_WIDTH`
`arg`	`s32*`

Valid argument values are as follows.

Value	Description
`-1`	Retrieve the current setting.
`DSI6C500K_DATA_WIDTH_16`	This option selects 16-bits of resolution.
`DSI6C500K_DATA_WIDTH_18`	This option selects 18-bits of resolution.
`DSI6C500K_DATA_WIDTH_20`	This option selects 20-bits of resolution.
`DSI6C500K_DATA_WIDTH_24`	This option selects 24-bits of resolution.

4.7.22. DSI6C500K_IOCTL_DIO_DIR_OUT

This service sets the Digital I/O Port pins as either input or output. The direction is adjustable in nibbles only.

Usage

Argument	Description
`request`	`DSI6C500K_IOCTL_DIO_DIR_OUT`
`arg`	`s32*`

Valid argument values are as follows.

Value	Description
`-1`	Retrieve the current setting.
`DSI6C500K_DIO_DIR_OUT_NONE`	All eight bits are inputs.
`DSI6C500K_DIO_DIR_OUT_0_3`	Bits zero to three are output and four to seven are inputs.
`DSI6C500K_DIO_DIR_OUT_4_7`	Bits four to seven are output and zero to three are inputs.
`DSI6C500K_DIO_DIR_OUT_0_7`	Bits zero to seven are output.

4.7.23. DSI6C500K_IOCTL_DIO_READ

This service reads the value of the Digital I/O Port pins. If a pin is configured as an output the value returned is the output value. If a pin is configured as an input the value returned in the value on the pin at the cable interface. Bit D0 refers to port pin zero.

Usage

Argument	Description
`request`	`DSI6C500K_IOCTL_DIO_READ`
`arg`	`s32*`

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

4.7.24. DSI6C500K_IOCTL_DIO_WRITE

This service writes to the Digital I/O Port pins.

Usage

Argument	Description
`request`	`DSI6C500K_IOCTL_DIO_WRITE`
`arg`	`s32*`

Valid argument values are from zero to `0xFF`, and -1. A value of -1 will return the current setting. Writes to output pins appear immediately at the cable interface. Writes to input pins are latched and will appear when the pin is subsequently configured as an output. Bit D0 refers to port pin zero.

4.7.25. DSI6C500K_IOCTL_EXT_CLK_SRC

This service configures the source for the external clock output.

Usage

Argument	Description
`request`	`DSI6C500K_IOCTL_EXT_CLK_SRC`
`arg`	`s32*`

Valid argument values are as follows.

Value	Description
`-1`	Retrieve the current setting.
`DSI6C500K_EXT_CLK_SRC_GEN`	This option selects the internal Rate Generator.
`DSI6C500K_EXT_CLK_SRC_GRP_0`	This option selects the Channel Group 0 sample clock.

4.7.26. DSI6C500K_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. The driver waits for initialization to complete before returning.

Usage

Argument	Description
`request`	`DSI6C500K_IOCTL_INITIALIZE`
`arg`	Not used.

4.7.27. DSI6C500K_IOCTL_IRQ_SEL

This service selects which firmware interrupt source may generate an interrupt.

Usage

Argument	Description
`request`	`DSI6C500K_IOCTL_IRQ_SEL`
`arg`	`s32*`

Valid argument values are as follows.

Value	Description
`-1`	Retrieve the current setting.
`DSI6C500K_IRQ_AI_BUF_THR_H2L`	This refers to a high-to-low transition of the input buffer threshold flag.
`DSI6C500K_IRQ_AI_BUF_THR_L2H`	This refers to a low-to-high transition of the input buffer threshold flag.
`DSI6C500K_IRQ_AI_BURST_DONE`	This refers to completion of a burst operation.
`DSI6C500K_IRQ_AUTOCAL_DONE`	This refers to Autocalibration completion.
`DSI6C500K_IRQ_CHAN_READY`	This refers to assertion of the Channel Ready status.
`DSI6C500K_IRQ_INIT_DONE`	This refers to completion of an initialization operation.

4.7.28. DSI6C500K_IOCTL_NDIV

This service configures the internal rate generator’s NDIV value.

Usage

Argument	Description
`request`	`DSI6C500K_IOCTL_NDIV`
`arg`	`s32*`

Valid argument values are from one to four, and -1. The value -1 returns the current setting.

4.7.29. DSI6C500K_IOCTL_NREF

This service configures the internal rate generator’s NREF value.

Usage

Argument	Description
`request`	`DSI6C500K_IOCTL_NREF`
`arg`	`s32*`

Valid argument values are from 25 to 300, and -1. The value -1 returns the current setting.

4.7.30. DSI6C500K_IOCTL_NVCO

This service configures the internal rate generator’s NVCO value.

Usage

Argument	Description
`request`	`DSI6C500K_IOCTL_NVCO`
`arg`	`s32*`

Valid argument values are from 25 to 300, and -1. The value -1 returns the current setting.

4.7.31. DSI6C500K_IOCTL_OSR

This service selects the analog input over sampling rate.

Usage

Argument	Description
`request`	`DSI6C500K_IOCTL_OSR`
`arg`	`s32*`

Valid argument values are as follows.

Value	Description
`-1`	Retrieve the current setting.
`DSI6C500K_OSR_32`	This refers to the 32x over sampling rate.
`DSI6C500K_OSR_64`	This refers to the 64x over sampling rate.
`DSI6C500K_OSR_128`	This refers to the 128x over sampling rate.
`DSI6C500K_OSR_256`	This refers to the 256x over sampling rate.

4.7.32. DSI6C500K_IOCTL_QUERY

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

Usage

Argument	Description
`request`	`DSI6C500K_IOCTL_QUERY`
`arg`	`s32*`

Valid argument values are as follows.

Value	Description
`DSI6C500K_QUERY_AUTOCAL_MS`	This returns the maximum duration of the Autocalibration cycle in milliseconds.
`DSI6C500K_QUERY_CHANNEL_MAX`	This returns the maximum number of input channels supported by all boards of the same model as the board accessed.
`DSI6C500K_QUERY_CHANNEL_QTY`	This returns the actual number of input channels on the current board.
`DSI6C500K_QUERY_COUNT`	This returns the number of query options supported by the IOCTL service.
`DSI6C500K_QUERY_DEVICE_TYPE`	This returns the identifier value for the board's type. The value should equal `GSC_DEV_TYPE_24DSI6C500K`.
`DSI6C500K_QUERY_FCLK_DEFAULT`	This gives the Burst Trigger Timer's default reference frequency (FCLK) in hertz.
`DSI6C500K_QUERY_FGEN_MAX`	This returns the maximum rate generator output (FGEN) in hertz.
`DSI6C500K_QUERY_FGEN_MIN`	This returns the minimum rate generator output (FGEN) in hertz.
`DSI6C500K_QUERY_FIFO_SIZE`	This returns the size of the input buffer in samples.
`DSI6C500K_QUERY_FILTER_FREQ`	This returns the installed filter frequency in hertz. The value zero is returned if no filter is installed and -1 is returned if the filter frequency is not known to the driver.
`DSI6C500K_QUERY_FREF_DEFAULT`	This gives the default sampling reference frequency (FREF) in hertz.
`DSI6C500K_QUERY_FSAMP_MAX`	This gives the maximum sample rate (FSAMP) in S/S.
`DSI6C500K_QUERY_FSAMP_MIN`	This gives the minimum sample rate (FSAMP) in S/S.

`DSI6C500K_QUERY_INIT_MS`	This returns the duration of a board initialization in milliseconds.
`DSI6C500K_QUERY_NDIV_MAX`	This returns the maximum supported NDIV value.
`DSI6C500K_QUERY_NDIV_MIN`	This returns the minimum supported NDIV value.
`DSI6C500K_QUERY_NREF_MAX`	This returns the maximum supported NREF value.
`DSI6C500K_QUERY_NREF_MIN`	This returns the minimum supported NREF value.
`DSI6C500K_QUERY_NVCO_MAX`	This returns the maximum supported NVCO value.
`DSI6C500K_QUERY_NVCO_MIN`	This returns the minimum supported NVCO value.
`DSI6C500K_IOCTL_SYNC_SRC`	This indicates the board's support for the SYNC Source selection feature.
`DSI6C500K_QUERY_V_RANGE`	This returns the board's factory configured voltage range. See the options values below.

The values returned for the `DSI6C500K_QUERY_FILTER_FREQ` query option are as follows.

Value	Description
`-1`	The filter frequency is unknown to the driver.
`0`	A filter is not installed.
`2,000,000`	The filter frequency is 2MHz.

The values returned for the `DSI6C500K_QUERY_V_RANGE` query option are as follows.

Value	Description
`DSI6C500K_QUERY_V_RANGE_2_5`	The board supports the voltage ranges of ±2.5 volts.
`DSI6C500K_QUERY_V_RANGE_5`	The board supports the voltage ranges of ±5 volts.
`DSI6C500K_QUERY_V_RANGE_6`	The board supports the voltage ranges of ±6 volts.
`DSI6C500K_QUERY_V_RANGE_10`	The board supports the voltage ranges of ±10 volts.

4.7.33. DSI6C500K_IOCTL_REG_MOD

This service performs a read-modify-write of a 24DSI6C500K register. This includes only the GSC firmware registers. The PCI and PLX Feature Set Registers are read-only. Refer to `24dsi6c500k.h` for a complete list of the GSC firmware registers.

Usage

Argument	Description
`request`	`DSI6C500K_IOCTL_REG_MOD`
`arg`	`gsc_reg_t*`

Definition

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

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

4.7.34. DSI6C500K_IOCTL_REG_READ

This service reads the value of a 24DSI6C500K register. This includes the PCI registers, the PLX Feature Set Registers and the GSC firmware registers. Refer to `24dsi6c500k.h` and `gsc_pci9056.h` for the complete list of accessible registers.

Usage

Argument	Description
`request`	`DSI6C500K_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.

4.7.35. DSI6C500K_IOCTL_REG_WRITE

This service writes a value to a 24DSI6C500K register. This includes only the GSC firmware registers. The PCI and PLX Feature Set Registers are read-only. Refer to `24dsi6c500k.h` for a complete list of the GSC firmware registers.

Usage

Argument	Description
`request`	`DSI6C500K_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.

4.7.36. DSI6C500K_IOCTL_RX_IO_ABORT

This service aborts an ongoing `read()` request. The service will wait for up to the read I/O timeout period for the request to complete.

Usage

Argument	Description
`request`	`DSI6C500K_IOCTL_RX_IO_ABORT`
`arg`	`s32*`

The results are reported as one of the following values.

Value	Description
`DSI6C500K_IO_ABORT_NO`	A `read()` request was not aborted.
`DSI6C500K_IO_ABORT_YES`	An ongoing `read()` request was aborted.

4.7.37. DSI6C500K_IOCTL_RX_IO_MODE

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

> **NOTE:** Applications may experience improved responsiveness with read requests by coordinating the Buffer Threshold with the number of samples requested. Refer to the `DSI6C500K_IOCTL_AIN_BUF_THRESH` service of section 4.7.5 on page 25.

Usage

Argument	Description
`request`	`DSI6C500K_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.

4.7.38. DSI6C500K_IOCTL_RX_IO_OVERFLOW

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

Usage

Argument	Description
`request`	`DSI6C500K_IOCTL_RX_IO_OVERFLOW`
`arg`	`s32*`

Valid argument values are as follows.

Value	Description
`-1`	Retrieve the current setting.

`DSI6C500K_IO_OVERFLOW_CHECK`	This option specifies that the check be performed.
`DSI6C500K_IO_OVERFLOW_IGNORE`	This option specifies that the check not be performed.

4.7.39. DSI6C500K_IOCTL_RX_IO_TIMEOUT

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

Usage

Argument	Description
`request`	`DSI6C500K_IOCTL_RX_IO_TIMEOUT`
`arg`	`s32*`

Valid argument values are in the range from zero to `3600`, `-1`, and `DSI6C500K_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 `DSI6C500K_IO_TIMEOUT_INFINITE` is used, then the driver will wait indefinitely rather than timing out. The default is `10` seconds.

4.7.40. DSI6C500K_IOCTL_RX_IO_UNDERFLOW

This service configures the read service to check for a data buffer underflow before performing read operations. Indeterminate data is returned when there is an underflow

Usage

Argument	Description
`request`	`DSI6C500K_IOCTL_RX_IO_UNDERFLOW`
`arg`	`s32*`

Valid argument values are as follows.

Value	Description
`-1`	Retrieve the current setting.
`DSI6C500K_IO_UNDERFLOW_CHECK`	This option specifies that the check be performed.
`DSI6C500K_IO_UNDERFLOW_IGNORE`	This option specifies that the check not be performed.

4.7.41. DSI6C500K_IOCTL_SYNC_SRC

This service configures the SYNC Source selection.

Usage

Argument	Description
`request`	`DSI6C500K_IOCTL_SYNC_SRC`
`arg`	`s32*`

Valid argument values are as follows.

Value	Description
`-1`	Retrieve the current setting. This is returned if the feature is unsupported.
`DSI6C500K_SYNC_SRC_EXT`	This selects the external SYNC Input from the cable interface.
`DSI6C500K_SYNC_SRC_G0_CA`	This configures the SYNC source per the channel Group-0 Clock Assignment.

4.7.42. DSI6C500K_IOCTL_SW_SYNC

This service initiates an ADC sync operation and, if in initiator mode, also generates an external sync output. The result of issuing a sync is dependent on the `DSI6C500K_IOCTL_SW_SYNC_MODE` setting (section 4.7.43, page 39). When initiating this operation, it is the application's responsibility to wait for the Channel Ready bit to be asserted.

Usage

Argument	Description
`request`	`DSI6C500K_IOCTL_SW_SYNC`
`arg`	Not used.

4.7.43. DSI6C500K_IOCTL_SW_SYNC_MODE

This service sets the context of the Software Sync operation.

Usage

Argument	Description
`request`	`DSI6C500K_IOCTL_SW_SYNC_MODE`
`arg`	`s32*`

Valid argument values are as follows.

Value	Description
`-1`	Retrieve the current setting.
`DSI6C500K_SW_SYNC_MODE_CLR_BUF`	This option causes a sync to clear the input buffer when there is a Software Sync request.
`DSI6C500K_SW_SYNC_MODE_SYNC`	Synchronize input channel scanning when there is a Software Sync request.

4.7.44. DSI6C500K_IOCTL_WAIT_CANCEL

This service resumes all threads blocked via the `DSI6C500K_IOCTL_WAIT_EVENT` IOCTL service (section 4.7.45, page 40), 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`	`DSI6C500K_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 4.7.45.2 on page 41.
`gsc`	This specifies the set of `DSI6C500K_WAIT_GSC_*` events whose wait requests are to be cancelled. Refer to section 4.7.45.3 on page 41.
`alt`	This is unused by the 24DSI6C500K driver and should be zero.
`io`	This specifies the set of `GSC_WAIT_IO_*` events whose wait requests are to be cancelled. Refer to section 4.7.45.4 on page 42.
`timeout_ms`	This is unused by wait cancel operations.
`count`	Upon return this indicates the number of waits that were cancelled.

4.7.45. DSI6C500K_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`	`DSI6C500K_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 4.7.45.1on page 41.

`main`	This specifies any number of `GSC_WAIT_MAIN_*` events that the thread is to wait for. Refer to section 4.7.45.2 on page 41.
`gsc`	This specifies any number of `DSI6C500K_WAIT_GSC_*` events that the thread is to wait for. Refer to section 4.7.45.3 on page 41.
`alt`	This is unused by the 24DSI6C500K driver and must be zero.
`io`	This specifies any number of `GSC_WAIT_IO_*` events that the thread is to wait for. Refer to section 4.7.45.4 on page 42.
`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.

4.7.45.1. `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.

4.7.45.2. `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 24DSI6C500K 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 24DSI6C500K.
`GSC_WAIT_MAIN_PCI`	This refers to any interrupt generated by the 24DSI6C500K.
`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.

4.7.45.3. `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 `DSI6C500K_IOCTL_IRQ_SEL` (section 4.7.27, page 32).

Value	**Description**
`DSI6C500K_WAIT_GSC_AI_BUF_THR_H2L`	This refers to a high-to-low transition of the input buffer threshold flag.
`DSI6C500K_WAIT_GSC_AI_BUF_THR_L2H`	This refers to a low-to-high transition of the input buffer threshold flag.
`DSI6C500K_WAIT_GSC_AI_BURST_DONE`	This refers to completion of an input burst operation.
`DSI6C500K_WAIT_GSC_AUTOCAL_DONE`	This refers to Autocalibration completion.
`DSI6C500K_WAIT_GSC_CHAN_READY`	This refers to assertion of the Channel Ready status.
`DSI6C500K_WAIT_GSC_INIT_DONE`	This refers to initialization completion.

4.7.45.4. `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 read requests.

Fields	**Description**
`GSC_WAIT_IO_RX_ABORT`	This refers to read requests which have been aborted.
`GSC_WAIT_IO_RX_DONE`	This refers to read requests which have been satisfied.
`GSC_WAIT_IO_RX_ERROR`	This refers to read requests which end due to an error.
`GSC_WAIT_IO_RX_TIMEOUT`	This refers to read requests which end due to the timeout period lapse.

4.7.46. DSI6C500K_IOCTL_WAIT_STATUS

This service counts all threads blocked via the `DSI6C500K_IOCTL_WAIT_EVENT` IOCTL service (section 4.7.45, page 40), 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`	`DSI6C500K_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 4.7.45.2 on page 41.
`gsc`	This specifies the set of `DSI6C500K_WAIT_GSC_*` events whose wait requests are to be counted. Refer to section 4.7.45.3 on page 41.
`alt`	This is unused by the 24DSI6C500K driver and should be zero.
`io`	This specifies the set of `GSC_WAIT_IO_*` events whose wait requests are to be counted. Refer to section 4.7.45.4 on page 42.
`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.

4.7.47. DSI6C500K_IOCTL_XCVR_TYPE

This service configures the transceiver type for the digital interface signals.

Usage

Argument	**Description**
`request`	`DSI6C500K_IOCTL_XCVR_TYPE`
`arg`	`s32*`

Valid argument values are as follows.

Value	**Description**
`-1`	Retrieve the current setting.
`DSI6C500K_XCVR_TYPE_LVDS`	This option selects LVDS signaling.
`DSI6C500K_XCVR_TYPE_TTL`	This option selects 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	`24dsi6c500k.h`	
Driver File	`24dsi6c500k.ko` † `24dsi6c500k.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 24DSI6C500K 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 `24dsi6c500k` 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/24dsi6c500k.*
```

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/24dsi6c500k/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/24dsi6c500k` 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/24dsi6c500k
```

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/24dsi6c500k` 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 24dsi6c500k
```

2. Verify that the driver module has been unloaded by issuing the below command. The module name `24dsi6c500k` 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	`24dsi6c500k_dsl.h`	…`/include/`
Library File	`24dsi6c500k_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
```

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 `dsi6c500k_open()` there is the utility file `open.c` containing the utility function `dsi6c500k_open_util()`. The naming pattern is as follows: API function `dsi6c500k_xxxx()`, utility file name `xxxx.c`, utility function `dsi6c500k_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 `DSI6C500K_IOCTL_QUERY` there is the utility file `util_query.c` containing the utility function `dsi6c500k_query()`. The naming pattern is as follows: IOCTL code `DSI6C500K_IOCTL_XXXX`, utility file name `util_xxxx.c`, utility function `dsi6c500k_xxxx()`.

7.1. Files

The utility files are summarized in the table below.

Description	Files	Location
Source Files	`*.c`, `*.h` …	…`/utils/`
Header File	`24dsi6c500k_utils.h`	…`/include/`

Library Files	`24dsi6c500k_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 24DSI6C500K. 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	`dsi6c500k_reg_list()`	Source File
Source File	`util_reg.c`	`…/utils/`
Header File	`24dsi6c500k_utils.h`	`…/include/`
Library File	`24dsi6c500k_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	`dsi6c500k_config_ai()`	Source File
Source File	`util_config_ai.c`	`…/utils/`
Header File	`24dsi6c500k_utils.h`	`…/include/`
Library File	`24dsi6c500k_utils.a`	`…/lib/`

8.3. 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.

8.3.1. PIO - Programmed I/O

In this mode data is transferred using repetitive register accesses. This is most applicable for low throughput requirements or for small transfer requests. The driver 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.

8.3.2. BMDMA - Block Mode DMA

For Block Mode DMA the driver initiates DMA transfers only after a sufficient volume of data has been received into the input buffer. After that amount of data is in the input buffer the driver initiates a DMA then sleeps until the DMA Done interrupt is received. Using this DMA mode, a user request is typically satisfied via a number of smaller DMA transfers.

8.3.3. DMDMA - Demand Mode DMA

This DMA mode is similar to the Block Mode, except that the DMA transfer is initiated immediately. Here however, the actual movement of data occurs as the data becomes available in the input buffer instead of after it has been received. Using this DMA mode, a user request is divided into smaller DMA transfers only if the request exceeds the size of the driver’s transfer buffer.

8.4. Multi-Board Synchronization

Multi-board synchronization is a feature of the 24DSI6C500K that enables two or more boards to sample analog input data in lock-step. Exercising this feature requires the boards to operate synchronously from the same clock source. This is done using the clock and sync signals on the cable interface. Though there are numerous varying ways of configuring the boards and of wiring the signals, the two basic configurations are described below.

8.4.1. Star Configuration

The *star* configuration generally permits all boards in the setup to operate with the least possible phase shift from one board to the next. This is accomplished by configuring all the boards in an identical manner and by wiring the clock and sync signals so that they follow as identical a path as possible from the initiator’s output to the input of the initiator and the targets. If there are three or more boards in the setup, then the clock and sync signal must go directly from the initiator’s output to a Clock Driver board, as illustrated in Figure 2. If there are only two boards in the setup, then a Clock Driver board is not needed, as illustrated in Figure 3. The table below shows the board programming that is specific to the *star* configuration.

Setting	**Initiator**	**Target(s)**
Initiator Mode	Initiator	Initiator
Rate Group 0 Clock Source	Direct External	Direct External
External Clock Output Source	Group 0	Group 0

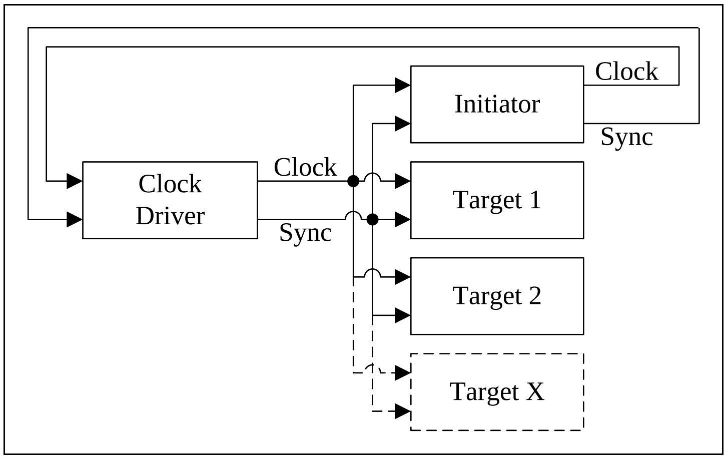

Figure 2 The *star* configuration with three or more boards requires a Clock Driver board.

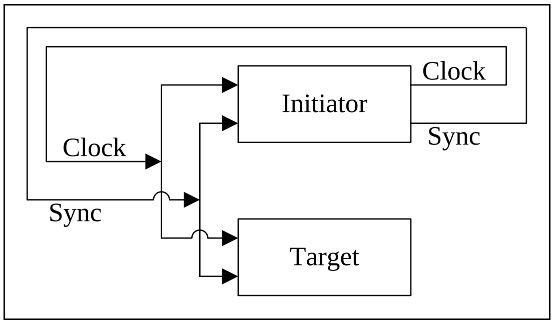

Figure 3 The *star* configuration with only two boards does not require a Clock Driver board.

8.4.2. Daisy Chain Configuration

The *daisy chain* configuration generally permits the most flexible placement of boards and wiring, and does not require a Clock Driver board. This is accomplished by configuring the boards and the wiring so that the clock and sync signals go from the initiator to the first target, then sequentially from the first target to the second and so on. This setup is applicable for any number of boards, as illustrated in Figure 4. The table below shows the board programming that is specific to the *daisy chain* configuration.

Setting	Initiator	Target(s)
Initiator Mode	Initiator	Target
Rate Group 0 Clock Source	Rate Generator A	Direct External
External Clock Output Source	Group 0	N/A (signals are passed through automatically)

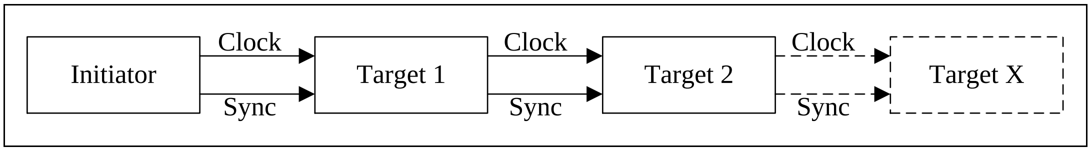

Figure 4 In this configuration the clock and sync signals are daisy chained from one board to the next.

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. din - Digital Input - …/din/

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

9.2. dout - Digital Output - …/dout/

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

9.3. fsamp - Sample Rate - …/fsamp/

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

9.4. id - Identify Board - …/id/

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

9.5. regs - Register Access - …/regs/

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

9.6. rxrate - Receive Rate - …/rxrate/

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

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

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

9.8. signals - Digital Signals - …/signals/

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

9.9. stream - Stream Rx Data to Disk - …/stream/

This application uses multiple threads with an intermediate buffer manager to stream data from the device to a data file. Numerous options are available for measuring performance of device reads, disk writes and buffer handling. Refer to the application file `readme.txt` for example information.

9.10. sw_sync - Software Sync - …/sw_sync/

This application repetitively activates the board's Software Sync feature and drives the output clock signal for a user specified period of time. This is done to facilitate setup of test equipment to capture those signals during actual use.

Document History

Revision	**Description**
July 8, 2024	Updated to version 2.4.111.50.0. Numerous, minor editorial changes. Updated the kernel support table. Updated the description of the Input Buffer Clear service. Updated the description of the Autocalibration service. Renamed all AUTO_CALIBRATE content to AUTOCAL. Renamed all AUTO_CAL content to AUTOCAL.
April 21, 2023	Updated to version 2.3.104.47.1. Corrected typo in the Over Sampling service.
April 3, 2023	Updated to version 2.3.104.47.0. Updated the kernel support table. Added section on environment variables. Updated the information for the open and close calls. Minor editorial updates. Updated the description of the Input Buffer Clear service.
February 28, 2022	Updated to version 2.2.97.38.0. Updated the kernel support table. Expanded automatic startup information. Removed statements about buffer enabling and clearing not being timed to scan boundaries. Removed statement about the buffer clear service also clearing the overflow and underflow status.
January 10, 2020	Updated to version 2.1.90.30.0. Updated the inside cover page. Updated the CPU and kernel support section. Updated Block Mode DMA macro and associated information. Added a licensing subsection. Added WAIT_EVENT note. Overhauled document.
December 8, 2016	Updated to version 2.0.69.18.0. Updated the device node name to include a period before the device index. Removed the `built` field from the `/proc` file. Updated the kernel support table. 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. Added support for infinite I/O timeouts. Updated the operating information section. Made various miscellaneous updates. Some document reorganization.
June 19, 2014	Updated to version 1.1.53.0. Added the `DSI6C500K_IOCTL_SYNC_SRC` IOCTL service and the `DSI6C500K_QUERY_SYNC_SRC` query option.
February 20, 2014	Initial release, version 1.0.52.0.