24DSI64C200K

24-bit, 64/48/32 channel, 250KS/S/Ch Delta-Sigma A/D Input

PCIe-24DSI64C200K

Linux Device Driver And API Library User Manual

Manual Revision: April 21, 2023
Driver Release Version 1.5.103.46.1

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 24DSI64C200K 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 24DSI64C200K 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
DMA	Direct Memory Access
DMDMA	Demand Mode DMA
GSC	General Standards Corporation
PCI	Peripheral Component Interconnect
PCIe	PCI Express
PIO	Programmed I/O
PMC	PCI Mezzanine Card

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 24DSI64C200K installation directory or any of its subdirectories.
24DSI64C200K	This is used as a general reference to any board supported by this driver.
API Library	This refers to the library implementing the 24DSI64C200K API.
Application	This is a user mode process, which runs in user space with user mode privileges.
Driver	This is the 24DSI64C200K 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 24DSI64C200K applications. The overall architecture is illustrated in Figure 1 below.

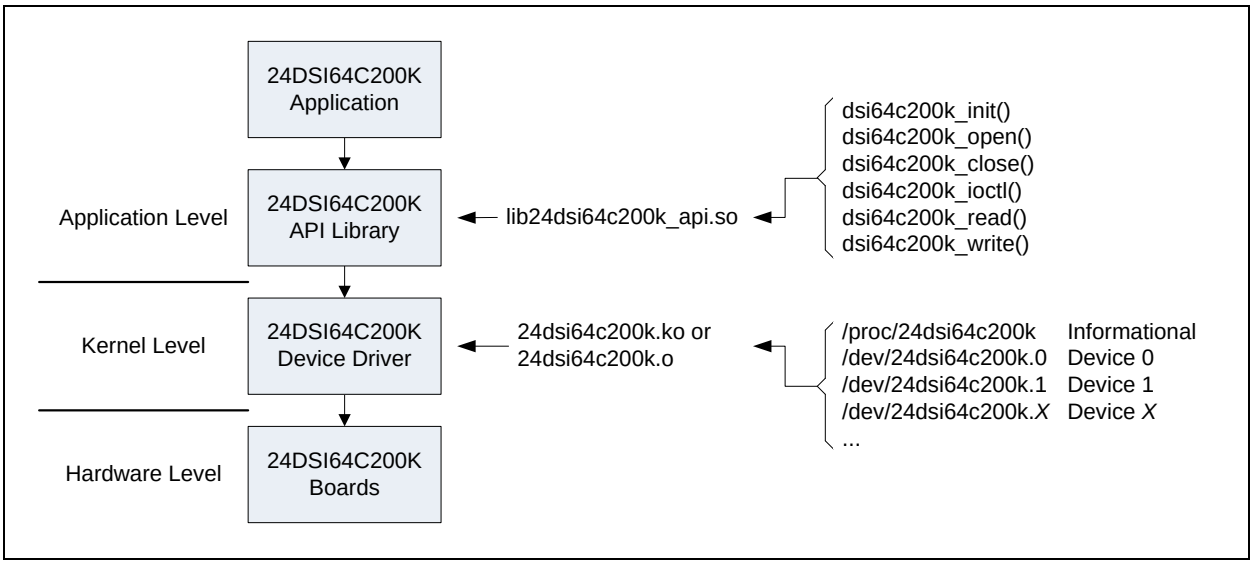

Figure 1 Basic architectural representation.

1.4.2. API Library

The primary means of accessing 24DSI64C200K boards is via the 24DSI64C200K API Library. This library forms a very thin layer between the application and the driver. Additional information is given in section 4 beginning on 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 24DSI64C200K 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 24DSI64C200K is a high-performance, 24-bit analog input board that contains either 64, 48 or 32 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 250K samples per second over each channel. Internal clocking permits sampling rates from 250K samples per second down to 33 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 24DSI64C200K to sustain continuous throughput from the cable interface independent of the PCI bus interface. The 24DSI64C200K also permits multiple boards to be synchronized so that all boards sample data in unison. In addition, the board includes auto-calibration capability and four general purpose digital I/O lines.

1.6. Reference Material

The following reference material may be of particular benefit in using the 24DSI64C200K. The specifications provide the information necessary for an in depth understanding of the specialized features implemented on this board.

- The applicable *24DSI64C200K 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 shipped 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/24dsi64c200k` file will be "`no`".

2.2. The /proc/ File System

While the driver is running, the text file `/proc/24dsi64c200k` can be read to obtain information about the driver. Each file entry includes an entry name followed immediately by a colon, a space character, and the entry value. Below is an example of what appears in the file, followed by descriptions of each entry.

```
version: 1.5.103.46
32-bit support: yes
boards: 1
models: 24DSI64C200K
```

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 lists the basic model names for the boards identified by the driver. There is one entry for each board. The order corresponds to that of the `/dev/24dsi64c200k.n` device nodes.

2.3. File List

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

File	Description
`24dsi64c200k.linux.tar.gz`	This archive contains the driver, the API Library and all related files.
`24dsi64c200k_linux_um.pdf`	This is a PDF version of this user manual, which is included in the archive.

2.4. Directory Structure

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

Directory	Content
`24dsi64c200k/`	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 24DSI64C200K API Library (section 4, page 17).
`…/docsrc/`	This directory contains the code samples from this document (section 6, page 48).
`…/driver/`	This directory contains the driver and its sources (section 5, page 44).
`…/include/`	This directory contains the include files for the various libraries.

…/lib/	This directory contains all of the libraries built from the driver archive.
…/samples/	This directory contains the sample applications (section 9, page 53).
…/utils/	This directory contains utility sources 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 `24dsi64c200k.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 `24dsi64c200k` in the current directory, and then copies all of the archive's files into this new directory.

   ```
   tar –xzvf 24dsi64c200k.linux.tar.gz
   ```

2.6. Removal

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

1. Shutdown the driver as described in section 5.6 on page 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 24dsi64c200k.linux.tar.gz 24dsi64c200k
   ```

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

   ```
   rm –f /dev/24dsi64c200k.*
   ```

5. If the automated 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 24DSI64C200K's `start` script. The file `rc.local` should be located in the `/etc/rc.d/` directory.

2.7. Overall Make Script

An Overall Make Script is included in the root installation directory. Executing this script will perform a make for all build targets included in the release, and it will also load the driver. The script is named `make_all`. Follow the below steps to perform an overall make and to load the driver.

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

1. Change to the driver root directory (…`/24dsi64c200k/`).

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

```
./make_all clean
```

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

```
./make_all
```

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/lib24dsi64c200k_api.so`
Defined and Not Empty	`==== Linking: ../lib/lib24dsi64c200k_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/24dsi64c200k_utils.a`
Defined and Not Empty	`==== Linking: ../lib/24dsi64c200k_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 24DSI64C200K 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 24DSI64C200K driver archive. For ease of use it is suggested that applications include only the single header file shown below rather than individually including those headers identified separately later in this document. Including this header file pulls in all other pertinent 24DSI64C200K specific header files. Therefore, sources may include only this one 24DSI64C200K header and make files may reference only this one 24DSI64C200K include directory.

Description	File	Location
Header File	`24dsi64c200k_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 24DSI64C200K driver archive. For ease of use it is suggested that applications link only the single library file shown below rather than individually linking those libraries identified separately later in this document. Linking this library file pulls in all other pertinent 24DSI64C200K specific static libraries. Therefore, make files may reference only this one 24DSI64C200K static library and only this one 24DSI64C200K library directory.

Description	File	Location
Static Library	`24dsi64c200k_main.a`	`…/lib/`

> **NOTE**: The 24DSI64C200K API Library is implemented as a shared library and is thus not linked with the 24DSI64C200K Main Library.

3.2.1. Build

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

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

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

   ```
   make clean
   ```

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

   ```
   make
   ```

3.2.2. System Libraries

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

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

4. API Library

The 24DSI64C200K API Library is the software interface between user applications and the 24DSI64C200K device driver. The interface is accessed by including the header file `24dsi64c200k_api.h`.

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

4.1. Files

The library source files are summarized in the table below.

File	Description
`api/*.c`	These are library source files.
`api/*.h`	These are library header files.
`api/makefile`	This is the library make file.
`api/makefile.dep`	This is an automatically generated make dependency file.
`include/24dsi64c200k_api.h`	This is the library interface header file.
`lib/lib24dsi64c200k_api.so`	This is the API Library shared library file. *

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

4.2. Build

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

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

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

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

   ```
   make clean
   ```

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

   ```
   make
   ```

4.3. Library Use

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

Description	File	Location	Linker Argument
Header File	`24dsi64c200k_api.h`	…`/include/`	
Shared Library	`lib24dsi64c200k_api.so`	…`/lib/`	
		`/usr/lib/`	`-l24dsi64c200k_api`

4.4. Macros

The API Library and driver interfaces include the following macros, which are defined in `24dsi64c200k.h`.

4.4.1. IOCTL

The IOCTL macros are documented in section 4.7 beginning on page 23.

4.4.2. Registers

The following gives the complete set of 24DSI64C200K registers.

4.4.2.1. GSC Registers

The following table gives the complete set of GSC specific 24DSI64C200K registers. For detailed definitions of these registers refer to the relevant *24DSI64C200K User Manual*. Please note that the set of registers supported by any given board may vary according to model and firmware version. For the set of supported registers and detailed definitions of these registers please refer to the appropriate *24DSI64C200K User Manual*.

Macro	Description
`DSI64C200K_GSC_AICR`	Analog Input Configuration Register
`DSI64C200K_GSC_ASIOCR`	Auxiliary SYNC I/O Control Register
`DSI64C200K_GSC_AVR`	Auto-Cal Values Register
`DSI64C200K_GSC_BBSR`	Burst Block Size Register
`DSI64C200K_GSC_BCFGR`	Board Configuration Register
`DSI64C200K_GSC_BCTLR`	Board Control Register
`DSI64C200K_GSC_BTTR`	Burst Trigger Timer Register
`DSI64C200K_GSC_BUFCR`	Buffer Control Register
`DSI64C200K_GSC_BUFSR`	Buffer Size Register
`DSI64C200K_GSC_DIOPR`	Digital I/O Port Register
`DSI64C200K_GSC_IDBR`	Input Data Buffer Register
`DSI64C200K_GSC_RCR`	Rate Control Register

4.4.2.2. PCI Configuration Registers

Access to the PCI registers is seldom required so these registers are not listed here. For the complete list of the PCI register identifiers refer to the driver header file `gsc_pci9056.h`, which is automatically included via `24dsi64c200k_api.h`.

4.4.2.3. PLX PCI9056 Feature Set Registers

Access to the PLX registers is seldom required so these registers are not listed here. For the complete list of the PLX register identifiers refer to the driver header file `gsc_pci9056.h`, which is automatically included via `24dsi64c200k_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 value of zero indicates success. A negative value indicates that the request could not be completed

successfully. The specific value returned is the negative of the corresponding error status value taken from `errno.h`. I/O services return positive values to indicate the number of bytes successfully transferred.

4.6.1. dsi64c200k_close()

This function is the entry point to close a connection to an open 24DSI64C200K board. The board is put in an initialized state before this call returns.

Prototype

```
int dsi64c200k_close(int fd);
```

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

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

Example

```
#include <stdio.h>

#include "24dsi64c200k_dsl.h"

int dsi64c200k_close_dsl(int fd)
{
    int errs;
    int ret;

    ret = dsi64c200k_close(fd);

    if (ret)
        printf("ERROR: dsi64c200k_close() returned %d\n", ret);

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

4.6.2. dsi64c200k_init()

This function is the entry point to initializing the 24DSI64C200K 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 dsi64c200k_init(void);
```

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

Example

```
#include <stdio.h>

#include "24dsi64c200k_dsl.h"

int dsi64c200k_init_dsl(void)
{
    int errs;
    int ret;

    ret = dsi64c200k_init();

    if (ret)
        printf("ERROR: dsi64c200k_init() returned %d\n", ret);

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

4.6.3. dsi64c200k_ioctl()

This function is the entry point to performing setup and control operations on a 24DSI64C200K board. This function should only be called after a successful open of the respective device. The specific operation performed varies according to the `request` argument. The `request` argument also governs the use and interpretation of the `arg` argument. The set of supported options for the `request` argument consists of the IOCTL services supported by the driver, which are defined in section 4.7 beginning on page 23.

Prototype

```
int dsi64c200k_ioctl(int fd, int request, void* arg);
```

Argument	Description
`fd`	This is the file descriptor of the device to access.
`request`	This specifies the desired operation to be performed.
`arg`	This is a `request` specific argument. Refer to the IOCTL services for additional information (section 4.7, page 23).

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

Example

```
#include <stdio.h>

#include "24dsi64c200k_dsl.h"

int dsi64c200k_ioctl_dsl(int fd, int request, void *arg)
{
    int errs;
    int ret;

    ret = dsi64c200k_ioctl(fd, request, arg);
```

```
    if (ret)
        printf("ERROR: dsi64c200k_ioctl() returned %d\n", ret);

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

4.6.4. dsi64c200k_open()

This function is the entry point to open a connection to a 24DSI64C200K board. The device is initialized before the function returns.

Prototype

```
int dsi64c200k_open(int device, int share, int* fd);
```

Argument	Description
device	This is the zero-based index of the 24DSI64C200K 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** -1 / There was an error. The device is not accessible. >= 0 / This is the handle to use to access the device in subsequent calls.

* If the index value is -1, then the API Library accesses /proc/24dsi64c200k.

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

Example

```
#include <stdio.h>

#include "24dsi64c200k_dsl.h"

int dsi64c200k_open_dsl(int device, int share, int* fd)
{
    int errs;
    int ret;

    ret = dsi64c200k_open(device, share, fd);

    if (ret)
        printf("ERROR: dsi64c200k_open() returned %d\n", ret);

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

4.6.4.1. Access Modes

Shared Access Mode:

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

Exclusive Access Mode:

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

4.6.5. dsi64c200k_read()

This function is the entry point to reading data from an open 24DSI64C200K. This function should only be called after a successful open of the respective device. The function reads up to `bytes` bytes from the board. The return value is the number of bytes actually read.

> **NOTE:** For additional information please refer to the I/O Modes section (section 8.2, page 50).

> **NOTE:** If an index of `-1` was passed to the `dsi64c200k_open()` call, then read requests will read from the text file `/proc/24dsi64c200k` (section 2.2, page 12).

Prototype

```
int dsi64c200k_read(int fd, void *dst, size_t bytes);
```

Argument	Description
`fd`	This is the file descriptor of the device to access.
`dst`	The data read will be put here.
`bytes`	This is the desired number of bytes to read. This must be a multiple of four (4).

Return Value	Description
`0` to `bytes`	The operation succeeded. A value less than `bytes` indicates that the request timed out.
`< 0`	An error occurred. This is the negative of `errno` from `errno.h`.

Example

```
#include <stdio.h>

#include "24dsi64c200k_dsl.h"

int dsi64c200k_read_dsl(int fd, void* dst, size_t bytes, size_t*
qty)
{
    int errs;
    int ret;

    ret = dsi64c200k_read(fd, dst, bytes);
```

```
    if (ret < 0)
        printf("ERROR: dsi64c200k_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 24DSI64C200K API Library and device driver implement the following IOCTL services. Each service is described along with the applicable `dsi64c200k_ioctl()` function arguments.

4.7.1. DSI64C200K_IOCTL_AI_BUF_CLEAR

This service clears the current content from the input buffer. This service waits for the firmware to complete the operation before returning, which can take up to three milliseconds at low sample rates. Before returning, this services clears both the Input Buffer Overflow and Input Buffer Underflow status bits.

Usage

Argument	Description
request	DSI64C200K_IOCTL_AI_BUF_CLEAR
arg	Not used.

4.7.2. DSI64C200K_IOCTL_AI_BUF_ENABLE

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

Usage

Argument	Description
request	DSI64C200K_IOCTL_AI_BUF_ENABLE
arg	s32*

Valid argument values are as follows.

Value	Description
-1	Retrieve the current setting.
DSI64C200K_AI_BUF_ENABLE_NO	This option disables input to the input buffer.
DSI64C200K_AI_BUF_ENABLE_YES	This option enables input to the input buffer.

4.7.3. DSI64C200K_IOCTL_AI_BUF_FILL_LVL

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

Usage

Argument	Description
request	DSI64C200K_IOCTL_AI_BUF_FILL_LVL
arg	s32*

Valid return values are from zero to `0x40000` (256K).

4.7.4. DSI64C200K_IOCTL_AI_BUF_OVERFLOW

This service operates on the Analog Input Overflow status.

Usage

Argument	Description
request	DSI64C200K_IOCTL_AI_BUF_OVERFLOW
arg	s32*

Valid argument values provided to the service are as follows.

Value	Description
-1	This option reports if an overflow has occurred.
DSI64C200K_AI_BUF_OVERFLOW_CLEAR	This option clears the overflow status.
DSI64C200K_AI_BUF_OVERFLOW_TEST	This option reports if an overflow has occurred.

Valid returned values are as follows.

Value	Description
DSI64C200K_AI_BUF_OVERFLOW_NO	An overflow did not occur.
DSI64C200K_AI_BUF_OVERFLOW_YES	An overflow did occur.

4.7.5. DSI64C200K_IOCTL_AI_BUF_THR_STS

This service reports the input buffer threshold status. The status is active (or asserted or set) while the buffer fill level exceeds the buffer threshold setting. The status is idle (or negated or clear) while the buffer fill level is equal to or below the buffer threshold setting.

Usage

Argument	Description
request	DSI64C200K_IOCTL_AI_BUF_THR_STS
arg	s32*

Valid returned values are as follows.

Value	Description
-1	Retrieve the current setting.
DSI64C200K_AI_BUF_THR_STS_ACTIVE	The threshold flag is set.
DSI64C200K_AI_BUF_THR_STS_IDLE	The threshold flag is not set.

4.7.6. DSI64C200K_IOCTL_AI_BUF_THRESH

This service sets the fill level at which the input buffer threshold status is asserted.

Usage

Argument	Description
request	DSI64C200K_IOCTL_AI_BUF_THRESH
arg	s32*

Valid argument values are from zero to `0x40000` (256K), and -1. A value of -1 will return the current threshold level setting.

4.7.7. DSI64C200K_IOCTL_AI_BUF_UNDERFLOW

This service operates on the Analog Input Underflow status.

Usage

Argument	Description
request	DSI64C200K_IOCTL_AI_BUF_UNDERFLOW
arg	s32*

Valid argument values provided to the service are as follows.

Value	Description
-1	Report if an underflow has occurred.
DSI64C200K_AI_BUF_UNDERFLOW_CLEAR	Clear the underflow status.
DSI64C200K_AI_BUF_UNDERFLOW_TEST	Report if an underflow has occurred.

Valid returned values are as follows.

Value	Description
DSI64C200K_AI_BUF_UNDERFLOW_NO	An underflow did not occur.
DSI64C200K_AI_BUF_UNDERFLOW_YES	An underflow did occur.

4.7.8. DSI64C200K_IOCTL_AI_FILTER

This service configures the analog input filter option.

Usage

Argument	Description
request	DSI64C200K_IOCTL_AI_FILTER
arg	s32*

Valid argument values are as follows.

Value	Description
-1	Retrieve the current setting.
DSI64C200K_AI_FILTER_LOW_LATENCY	This option selects the low latency filter.
DSI64C200K_AI_FILTER_WIDEBAND	This option selects the wideband filter.

4.7.9. DSI64C200K_IOCTL_AI_MODE

This service configures the analog input mode.

Usage

Argument	Description
request	DSI64C200K_IOCTL_AI_MODE
arg	s32*

Valid argument values are as follows.

Value	Description
-1	Retrieve the current setting.
DSI64C200K_AI_MODE_DIFF	This option selects differential operation.
DSI64C200K_AI_MODE_VREF	This option connects the input channels to the onboard VREF signal.
DSI64C200K_AI_MODE_ZERO	This option connects the input channels to the onboard zero voltage signal.

4.7.10. DSI64C200K_IOCTL_AUTO_CAL_STS

This service reports the auto-calibration status.

Usage

Argument	Description
request	DSI64C200K_IOCTL_AUTO_CAL_STS
arg	s32*

Valid argument values returned are as follows.

Value	Description
DSI64C200K_AUTO_CAL_STS_ACTIVE	An auto-calibration cycle is in progress.
DSI64C200K_AUTO_CAL_STS_FAIL	The most recent auto-calibration cycle failed.
DSI64C200K_AUTO_CAL_STS_PASS	The most recent auto-calibration cycle passed.

4.7.11. DSI64C200K_IOCTL_AUTO_CALIBRATE

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

Usage

Argument	Description
request	DSI64C200K_IOCTL_AUTO_CALIBRATE
arg	Not used.

4.7.12. DSI64C200K_IOCTL_AUX_CLOCK

This service configures the operation of the sample clock signal on the auxiliary connector.

Usage

Argument	Description
request	DSI64C200K_IOCTL_AUX_CLOCK
arg	s32*

Valid argument values are as follows.

Value	Description
-1	Retrieve the current setting.
DSI64C200K_AUX_CLOCK_INACTIVE	This option disables the signals operation.
DSI64C200K_AUX_CLOCK_INPUT	This option configures the signal as an input.
DSI64C200K_AUX_CLOCK_OUTPUT	This option configures the signal as an output.

4.7.13. DSI64C200K_IOCTL_AUX_SYNC

This service configures the operation of the SYNC signal on the auxiliary connector.

Usage

Argument	Description
request	DSI64C200K_IOCTL_AUX_SYNC
arg	s32*

Valid argument values are as follows.

Value	Description
-1	Retrieve the current setting.
DSI64C200K_AUX_SYNC_INACTIVE	This option disables the signals operation.
DSI64C200K_AUX_SYNC_INPUT	This option configures the signal as an input.
DSI64C200K_AUX_SYNC_OUTPUT	This option configures the signal as an output.

4.7.14. DSI64C200K_IOCTL_BURST_ENABLE

This service enables or disables input bursting.

Usage

Argument	Description
request	DSI64C200K_IOCTL_BURST_ENABLE
arg	s32*

Valid argument values are as follows.

Value	Description
-1	Retrieve the current setting.
DSI64C200K_BURST_ENABLE_NO	This option disables input bursting.
DSI64C200K_BURST_ENABLE_YES	This option enables input bursting.

4.7.15. DSI64C200K_IOCTL_BURST_RATE_DIV

This service adjusts the Burst Rate Divisor, which controls the Burst Timer rate.

Usage

Argument	Description
request	DSI64C200K_IOCTL_BURST_RATE_DIV
arg	s32*

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

4.7.16. DSI64C200K_IOCTL_BURST_SIZE

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

Usage

Argument	Description
request	DSI64C200K_IOCTL_BURST_SIZE
arg	s32*

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

4.7.17. DSI64C200K_IOCTL_BURST_TIMER

This service enables or disables the input bursting timer.

Usage

Argument	Description
request	DSI64C200K_IOCTL_BURST_TIMER
arg	s32*

Valid argument values are as follows.

Value	Description
-1	Retrieve the current setting.
DSI64C200K_BURST_TIMER_DISABLE	This option disables the input burst timer.
DSI64C200K_BURST_TIMER_ENABLE	This option enables the input burst timer.

4.7.18. DSI64C200K_IOCTL_BURST_TRIGGER

This service initiates a burst operation.

Usage

Argument	Description
request	DSI64C200K_IOCTL_BURST_TRIGGER
arg	Not used.

4.7.19. DSI64C200K_IOCTL_CHAN_GRP_ACTIVE

This service selects which of the eight channel groups are to be active during sampling operations. If a bit is set, then the corresponding channel group is enabled. If a bit is clear, then the corresponding channel group is disabled.

Usage

Argument	Description
request	DSI64C200K_IOCTL_CHAN_GRP_ACTIVE
arg	s32*

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

4.7.20. DSI64C200K_IOCTL_CHANNELS_READY

This service operates on the Channels Ready status.

Usage

Argument	Description
request	DSI64C200K_IOCTL_CHANNELS_READY
arg	s32*

Valid argument values provided to the service are as follows.

Value	Description
-1	This reports if the status is *ready*.
DSI64C200K_CHANNELS_READY_TEST	This reports if the status is *ready*.
DSI64C200K_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
DSI64C200K_CHANNELS_READY_NO	The status is *not ready*.
DSI64C200K_CHANNELS_READY_YES	The status is *ready*.

4.7.21. DSI64C200K_IOCTL_CLK_SRC

This service configures the input clocking source option.

Usage

Argument	Description
request	DSI64C200K_IOCTL_CLK_SRC
arg	s32*

Valid argument values are as follows.

Value	Description
-1	Retrieve the current setting.
DSI64C200K_CLK_SRC_EXT_FGEN	This replaces the internal FGEN signal with the external clock.
DSI64C200K_CLK_SRC_EXT_MCLK	This replaces the internal MCLK signal with the external clock.
DSI64C200K_CLK_SRC_RATE_GEN	This option selects the internal Rate Generator.

4.7.22. DSI64C200K_IOCTL_CONTROL_MODE

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

Usage

Argument	Description
request	DSI64C200K_IOCTL_CONTROL_MODE
arg	s32*

Valid argument values are as follows.

Value	Description
-1	Retrieve the current setting.
DSI64C200K_CONTROL_MODE_INITIATOR	This option selects initiator mode operation.
DSI64C200K_CONTROL_MODE_TARGET	This option selects target mode operation.

4.7.23. DSI64C200K_IOCTL_DATA_FORMAT

This service configures the data encoding format.

Usage

Argument	Description
request	DSI64C200K_IOCTL_DATA_FORMAT
arg	s32*

Valid argument values are as follows.

Value	Description
-1	Retrieve the current setting.
DSI64C200K_DATA_FORMAT_2S_COMP	This option selects the Twos Compliment data format.
DSI64C200K_DATA_FORMAT_OFF_BIN	This option selects the Offset Binary encoding format.

4.7.24. DSI64C200K_IOCTL_DATA_WIDTH

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

Usage

Argument	Description
request	DSI64C200K_IOCTL_DATA_WIDTH
arg	s32*

Valid argument values are as follows.

Value	Description
-1	Retrieve the current setting.
DSI64C200K_DATA_WIDTH_16	This option selects 16-bits of resolution.
DSI64C200K_DATA_WIDTH_18	This option selects 18-bits of resolution.
DSI64C200K_DATA_WIDTH_20	This option selects 20-bits of resolution.
DSI64C200K_DATA_WIDTH_24	This option selects 24-bits of resolution.

4.7.25. DSI64C200K_IOCTL_DIO_DIR_OUT

This service sets the Digital I/O Port pins as either input or output. If a bit is set, then the corresponding port pin is an output. If a bit is clear, then the corresponding port pin is an input.

Usage

Argument	Description
request	DSI64C200K_IOCTL_DIO_DIR_OUT
arg	s32*

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

4.7.26. DSI64C200K_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.

Usage

Argument	Description
request	DSI64C200K_IOCTL_DIO_READ
arg	s32*

Valid values returned are from zero to 0xF.

4.7.27. DSI64C200K_IOCTL_DIO_WRITE

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

Usage

Argument	Description
request	DSI64C200K_IOCTL_DIO_WRITE
arg	s32*

Valid argument values are from zero to 0xF, 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.

4.7.28. DSI64C200K_IOCTL_EXT_CLK_SRC

This service configures the source for the external clock output.

Usage

Argument	Description
request	DSI64C200K_IOCTL_EXT_CLK_SRC
arg	s32*

Valid argument values are as follows.

Value	Description
-1	Retrieve the current setting.
DSI64C200K_EXT_CLK_SRC_GEN	This option selects the internal Rate Generator.
DSI64C200K_EXT_CLK_SRC_MCLK	This option selects the internal MCLK signal.

4.7.29. DSI64C200K_IOCTL_FGEN_DIV

This service selects the FGEN divisor value used to generate the sample clock.

Usage

Argument	Description
request	DSI64C200K_IOCTL_FGEN_DIV
arg	s32*

Valid argument values are as follows.

Value	Description
-1	Retrieve the current setting.
DSI64C200K_FGEN_DIV_1	This leaves the FGEN signal as is.

DSI64C200K_FGEN_DIV_2	This divides the FGEN signal by two.
DSI64C200K_FGEN_DIV_4	This divides the FGEN signal by four.
DSI64C200K_FGEN_DIV_8	This divides the FGEN signal by eight.
DSI64C200K_FGEN_DIV_16	This divides the FGEN signal by 16.

4.7.30. DSI64C200K_IOCTL_INIT_ADCS

This service initializes the boards Analog-to-Digital Converters. The driver waits for initialization to complete before returning.

Usage

Argument	Description
request	DSI64C200K_IOCTL_INIT_ADCS
arg	Not used.

4.7.31. DSI64C200K_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	DSI64C200K_IOCTL_INITIALIZE
arg	Not used.

4.7.32. DSI64C200K_IOCTL_IRQ_SEL

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

Usage

Argument	Description
request	DSI64C200K_IOCTL_IRQ_SEL
arg	s32*

Valid argument values are as follows.

Value	Description
-1	Retrieve the current setting.
DSI64C200K_IRQ_AI_BUF_THR_H2L	This refers to a high-to-low transition of the input buffer threshold flag.
DSI64C200K_IRQ_AI_BUF_THR_L2H	This refers to a low-to-high transition of the input buffer threshold flag.
DSI64C200K_IRQ_AI_BURST_DONE	This refers to completion of a burst operation.
DSI64C200K_IRQ_AUTO_CAL_DONE	This refers to Auto-Calibration completion.
DSI64C200K_IRQ_CHAN_READY_H2L	This refers to assertion of the Channels Ready status.
DSI64C200K_IRQ_CHAN_READY_L2H	This refers to negating of the Channels Ready status.
DSI64C200K_IRQ_INIT_DONE	This refers to completion of an initialization operation.

4.7.33. DSI64C200K_IOCTL_MCLK_DIV

This service selects the MCLK divisor value, which is used to produce a desired sample rate.

Usage

Argument	Description
request	DSI64C200K_IOCTL_FGEN_DIV
arg	s32*

Valid argument values are as follows.

Value	Description
-1	Retrieve the current setting.
DSI64C200K_MCLK_DIV_4_FAST	This divides the MCLK signal by four.
DSI64C200K_MCLK_DIV_8_MEDIAN	This divides the MCLK signal by eight.
DSI64C200K_MCLK_DIV_32_ECO	This divides the MCLK signal by 32.

4.7.34. DSI64C200K_IOCTL_NREF

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

Usage

Argument	Description
request	DSI64C200K_IOCTL_NREF
arg	s32*

Valid argument values are from 20 to 300, and -1. The value -1 returns the current setting. The optimal range is from 25 to 100.

4.7.35. DSI64C200K_IOCTL_NVCO

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

Usage

Argument	Description
request	DSI64C200K_IOCTL_NVCO
arg	s32*

Valid argument values are from 20 to 300, and -1. The value -1 returns the current setting. The optimal range is from 25 to 100.

4.7.36. DSI64C200K_IOCTL_OVER_SAMPLE

This service selects the analog input over sampling rate, which is used to produce a desired sample rate.

Usage

Argument	Description
request	DSI64C200K_IOCTL_OVER_SAMPLE
arg	s32*

Valid argument values are as follows.

Value	Description
-1	Retrieve the current setting.
DSI64C200K_OVER_SAMPLE_32	This refers to the 32x over sampling rate.
DSI64C200K_OVER_SAMPLE_64	This refers to the 64x over sampling rate.
DSI64C200K_OVER_SAMPLE_128	This refers to the 128x over sampling rate.
DSI64C200K_OVER_SAMPLE_1024	This refers to the 1024x over sampling rate.

4.7.37. DSI64C200K_IOCTL_QUERY

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

Usage

Argument	Description
request	DSI64C200K_IOCTL_QUERY
arg	s32*

Valid argument values are as follows.

Value	Description
DSI64C200K_QUERY_AUTO_CAL_MS	This returns the maximum duration of the Auto Calibration cycle in milliseconds.
DSI64C200K_QUERY_CHANNEL_MAX	This returns the maximum number of input channels supported by all boards of the same model as the board accessed.
DSI64C200K_QUERY_CHANNEL_QTY	This returns the actual number of input channels on the current board.
DSI64C200K_QUERY_COUNT	This returns the number of query options supported by the IOCTL service.
DSI64C200K_QUERY_DEVICE_TYPE	This returns the identifier value for the board's type. The value should equal GSC_DEV_TYPE_24DSI64C200K.
DSI64C200K_QUERY_FGEN_MAX	This returns the maximum rate generator output (FGEN) in hertz.
DSI64C200K_QUERY_FGEN_MIN	This returns the minimum rate generator output (FGEN) in hertz.
DSI64C200K_QUERY_FIFO_SIZE	This returns the size of the input buffer in samples.
DSI64C200K_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.
DSI64C200K_QUERY_FREF_DEFAULT	This gives the default reference frequency (FREF) in hertz.
DSI64C200K_QUERY_FSAMP_MAX	This gives the maximum sample rate (FSAMP) in S/S.
DSI64C200K_QUERY_FSAMP_MIN	This gives the minimum sample rate (FSAMP) in S/S.
DSI64C200K_QUERY_FW_REV	This gives the firmware revision number.
DSI64C200K_QUERY_INIT_ADC_MS	This returns the duration of an ADC initialization in milliseconds.
DSI64C200K_QUERY_INIT_MS	This returns the duration of a board initialization in milliseconds.
DSI64C200K_QUERY_MASTER_CLOCK	This gives the frequency of the master clock used to configure the burst rate.
DSI64C200K_QUERY_MCLK_MAX	This gives the maximum supported MCLK frequency.
DSI64C200K_QUERY_MCLK_MIN	This gives the minimum supported MCLK frequency.
DSI64C200K_QUERY_NREF_MAX	This returns the maximum supported NREF value.

DSI64C200K_QUERY_NREF_MIN	This returns the minimum supported NREF value.
DSI64C200K_QUERY_NREF_OPT_MAX	This returns the maximum optimal NREF value.
DSI64C200K_QUERY_NREF_OPT_MIN	This returns the minimum optimal NREF value.
DSI64C200K_QUERY_NVCO_MAX	This returns the maximum supported NVCO value.
DSI64C200K_QUERY_NVCO_MIN	This returns the minimum supported NVCO value.
DSI64C200K_QUERY_NVCO_OPT_MAX	This returns the maximum optimal NVCO value.
DSI64C200K_QUERY_NVCO_OPT_MIN	This returns the minimum optimal NVCO value.
DSI64C200K_QUERY_V_RANGE	This returns the board's factory configured voltage range. See the option values below.

The values returned for the DSI64C200K_QUERY_V_RANGE query option are as follows.

Value	Description
-1	The voltage range is not recognized by the driver.
DSI64C200K_QUERY_V_RANGE_10	The board supports the voltage ranges of ±10 volts.

4.7.38. DSI64C200K_IOCTL_REG_MOD

This service performs a read-modify-write of a 24DSI64C200K register. This includes only the GSC firmware registers. The PCI and PLX Feature Set Registers are read-only. Refer to 24dsi64c200k.h for a complete list of the GSC firmware registers.

Usage

Argument	Description
request	DSI64C200K_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.

4.7.39. DSI64C200K_IOCTL_REG_READ

This service reads the value of a 24DSI64C200K register. This includes the PCI registers, the PLX Feature Set Registers and the GSC firmware registers. Refer to 24dsi64c200k.h and gsc_pci9056.h for the complete list of accessible registers.

Usage

Argument	Description
request	DSI64C200K_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 requests.

4.7.40. DSI64C200K_IOCTL_REG_WRITE

This service writes a value to a 24DSI64C200K register. This includes only the GSC firmware registers. The PCI and PLX Feature Set Registers are read-only. Refer to `24dsi64c200k.h` for a complete list of the GSC firmware registers.

Usage

Argument	Description
request	DSI64C200K_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 requests.

4.7.41. DSI64C200K_IOCTL_RX_IO_ABORT

This service aborts an ongoing `dsi64c200k_read()` request. The service will wait for up to the read I/O timeout period for the request to complete.

Usage

Argument	Description
request	DSI64C200K_IOCTL_RX_IO_ABORT
arg	s32*

The results are reported as one of the following values.

Value	Description
DSI64C200K_IO_ABORT_NO	A read request was not aborted.
DSI64C200K_IO_ABORT_YES	An ongoing read request was aborted.

4.7.42. DSI64C200K_IOCTL_RX_IO_MODE

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

Usage

Argument	Description
request	DSI64C200K_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.

4.7.43. DSI64C200K_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	DSI64C200K_IOCTL_RX_IO_OVERFLOW
arg	s32*

Valid argument values are as follows.

Value	Description
-1	Retrieve the current setting.
DSI64C200K_IO_OVERFLOW_CHECK	This option specifies that the check be performed.
DSI64C200K_IO_OVERFLOW_IGNORE	This option specifies that the check not be performed.

4.7.44. DSI64C200K_IOCTL_RX_IO_TIMEOUT

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

Usage

Argument	Description
request	DSI64C200K_IOCTL_RX_IO_TIMEOUT
arg	s32*

Valid argument values are in the range from zero to 3600, -1, and DSI64C200K_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 DSI64C200K_IO_TIMEOUT_INFINITE is used, then the driver will wait indefinitely rather than timing out. The default is 10 seconds.

4.7.45. DSI64C200K_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	DSI64C200K_IOCTL_RX_IO_UNDERFLOW
arg	s32*

Valid argument values are as follows.

Value	Description
-1	Retrieve the current setting.
DSI64C200K_IO_UNDERFLOW_CHECK	This option specifies that the check be performed.
DSI64C200K_IO_UNDERFLOW_IGNORE	This option specifies that the check not be performed.

4.7.46. DSI64C200K_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 DSI64C200K_IOCTL_SW_SYNC_MODE setting (refer to section 4.7.47 on page 38). When initiating this operation, it is the application's responsibility to wait for the Channels Ready bit to be asserted.

Usage

Argument	Description
request	DSI64C200K_IOCTL_SW_SYNC
arg	Not used.

4.7.47. DSI64C200K_IOCTL_SW_SYNC_MODE

This service sets the context of the Software Sync operation.

Usage

Argument	Description
request	DSI64C200K_IOCTL_SW_SYNC_MODE
arg	s32*

Valid argument values are as follows.

Value	Description
-1	Retrieve the current setting.
DSI64C200K_SW_SYNC_MODE_CLR_BUF	This option causes a sync to clear the input buffer when there is a Software Sync request.
DSI64C200K_SW_SYNC_MODE_SYNC	Synchronize input channel scanning when there is a Software Sync request.

4.7.48. DSI64C200K_IOCTL_SYNC_SRC

This service selects the source for signals generating a SYNC operation.

Usage

Argument	Description
request	DSI64C200K_IOCTL_SYNC_SRC
arg	s32*

Valid argument values are as follows.

Value	Description
-1	Retrieve the current setting.
DSI64C200K_SYNC_SRC_EXT_BCR	This option enables SYNC generation from either the external SYNC Input signal or the Synchronize Inputs bit in the Board Control Register.
DSI64C200K_SYNC_SRC_TIMER_BCR	This option enables SYNC generation from either the internal burst timer or the Synchronize Inputs bit in the Board Control Register.

4.7.49. DSI64C200K_IOCTL_WAIT_CANCEL

This service resumes all threads blocked via the DSI64C200K_IOCTL_WAIT_EVENT IOCTL service (section 4.7.50, 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	DSI64C200K_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 4.7.50.2 on page 41.
gsc	This specifies the set of DSI64C200K_WAIT_GSC_* events whose wait requests are to be cancelled. Refer to section 4.7.50.3 on page 41.
alt	This is unused by the 24DSI64C200K 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 4.7.50.4 on page 41.
timeout_ms	This is unused by wait cancel operations.
count	Upon return this indicates the number of waits that were cancelled.

4.7.50. DSI64C200K_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	DSI64C200K_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 4.7.50.1on page 41.
main	This specifies any number of GSC_WAIT_MAIN_* events that the thread is to wait for. Refer to section 4.7.50.2 on page 41.
gsc	This specifies any number of DSI64C200K_WAIT_GSC_* events that the thread is to wait for. Refer to section 4.7.50.3 on page 41.
alt	This is unused by the 24DSI64C200K 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 4.7.50.4 on page 41.
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.

4.7.50.1. 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.

4.7.50.2. 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 24DSI64C200K 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 24DSI64C200K.
GSC_WAIT_MAIN_PCI	This refers to any interrupt generated by the 24DSI64C200K.
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.

4.7.50.3. 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 DSI64C200K_IOCTL_IRQ_SEL (section 4.7.32, page 32).

Value	Description
DSI64C200K_WAIT_GSC_AI_BUF_THR_H2L	This refers to a high-to-low transition of the input buffer threshold flag.
DSI64C200K_WAIT_GSC_AI_BUF_THR_L2H`	This refers to a low-to-high transition of the input buffer threshold flag.
DSI64C200K_WAIT_GSC_AI_BURST_DONE	This refers to completion of an input burst operation.
DSI64C200K_WAIT_GSC_AUTO_CAL_DONE	This refers to Auto-Calibration completion.
DSI64C200K_WAIT_GSC_CHAN_READY_H2L	This refers to negation of the Channels Ready status.
DSI64C200K_WAIT_GSC_CHAN_READY_L2H	This refers to assertion of the Channels Ready status.
DSI64C200K_WAIT_GSC_INIT_DONE	This refers to initialization completion.

4.7.50.4. 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.

4.7.51. DSI64C200K_IOCTL_WAIT_STATUS

This service counts all threads blocked via the `DSI64C200K_IOCTL_WAIT_EVENT` IOCTL service (section 4.7.50, 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	DSI64C200K_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 4.7.50.2 on page 41.
gsc	This specifies the set of DSI64C200K_WAIT_GSC_* events whose wait requests are to be counted. Refer to section 4.7.50.3 on page 41.
alt	This is unused by the 24DSI64C200K 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 4.7.50.4 on page 41.
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.

4.7.52. DSI64C200K_IOCTL_XCVR_TYPE

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

Usage

Argument	Description
request	DSI64C200K_IOCTL_XCVR_TYPE
arg	s32*

Valid argument values are as follows.

Value	Description
-1	Retrieve the current setting.
DSI64C200K_XCVR_TYPE_LVDS	This option selects LVDS signaling.
DSI64C200K_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 source files are summarized in the table below.

File	Description
`driver/*.c`	The driver source files.
`driver/*.h`	The driver header files.
`driver/start`	Shell script to install the driver executable and device nodes.
`driver/24dsi64c200k.h`	This is the driver interface header file.
`driver/Makefile`	This is the driver make file.

5.2. Build

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

Follow the below steps to build the driver.

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

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

   ```
   make clean
   ```

3. Build the driver by issuing the below command.

   ```
   make
   ```

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

5.3. Startup

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

The startup script used in this procedure is designed to ensure that the driver module in the install directory is the module that is loaded. The currently loaded driver is first unloaded before attempting to load the module from the script's directory. The script also 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 correspond to the number of boards identified by the driver.

5.3.1. Manual Driver Startup Procedures

Start the driver manually by following the below listed steps.

NOTE: The following steps may require elevated privileges.

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

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

   ```
   ./start
   ```

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

   NOTE: The 24DSI64C200K 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 `24dsi64c200k` 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/24dsi64c200k.*
   ```

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/24dsi64c200k/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 you 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/24dsi64c200k` 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/24dsi64c200k
   ```

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/24dsi64c200k` 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 24dsi64c200k
   ```

2. Verify that the driver module has been unloaded by issuing the below command. The module name `24dsi64c200k` should not be in the listed output.

   ```
   lsmod
   ```

6. Document Source Code Examples

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

6.1. Files

The library files are summarized in the table below.

File	Description
`docsrc/*.c`	These are the C source files.
`docsrc/makefile`	This is the library make file.
`docsrc/makefile.dep`	This is an automatically generated make dependency file.
`include/24dsi64c200k_dsl.h`	This is the primary utility header file.
`lib/24dsi64c200k_dsl.a`	This is the statically linkable library file.

6.2. Build

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

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

2. Remove existing build targets 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 below listed header file in each source file using a component of the library interface. At link time include the below listed library file with the objects being linked with the application.

Description	File	Location
Header File	`24dsi64c200k_dsl.h`	…`/include/`
Static Link Library	`24dsi64c200k_dsl.a`	…`/lib/`

7. Utility Source Code

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

7.1. Files

The library files are summarized in the table below.

File	Description
`utils/util_*.c`	These are device specific utility source files.
`utils/gsc_*.c`	These are device and OS independent utility source files.
`utils/os_*.c`	These are OS specific utility source files.
`utils/makefile`	This is the library make file.
`utils/makefile.dep`	This is an automatically generated make dependency file.
`include/24dsi64c200k_utils.h`	This is the primary utility header file.
`lib/24dsi64c200k_utils.a`	This is the statically linkable library file.

7.2. Build

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

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

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

   ```
   make clean
   ```

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

   ```
   make
   ```

7.3. Library Use

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

Description	File	Location
Header File	`24dsi64c200k_utils.h`	…`/include/`
Static Link Libraries	`24dsi64c200k_utils.a`	…`/lib/`

8. Operating Information

This section explains some basic operational procedures for using the 24DSI64C200K. This is in no way intended to be a comprehensive guide. This is simply to address a very few issues relating to their use.

8.1. Analog Input Configuration

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

Item	Name/File	Location
Function	`dsi64c200k_config_ai()`	Source File
Source File	`util_config_ai.c`	`…/utils/`
Header File	`24dsi64c200k_utils.h`	`…/include/`
Library File	`24dsi64c200k_utils.a`	`…/lib/`

8.2. I/O Modes

All data read requests move the requested data from the board's input buffer, to an intermediate driver buffer, then from there to application memory. The data is processed in chunks no larger than the size of the transfer buffer. The transfer buffer size is typically the size of the input buffer, or larger, unless insufficient memory is available to the driver. The process used to move data from the input buffer to the intermediate buffer is according to the I/O mode selection.

8.2.1. PIO - Programmed I/O

In PIO mode the driver reads data by repetitive registers reads from the input data buffer register until either the request is satisfied or the I/O timeout expires, whichever occurs first.

8.2.2. 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.

8.2.3. 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.

8.3. Debugging Aids

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

8.3.1. 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/`

8.3.2. Detailed Register Dump

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

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

Description	File/Name	Location
Function	`dsi64c200k_reg_list()`	Source File
Source File	`util_reg.c`	`…/utils/`
Header File	`24dsi64c200k_utils.h`	`…/include/`
Library File	`24dsi64c200k_utils.a`	`…/lib/`

8.4. Multi-Board Synchronization

Multi-board synchronization is a feature of the 24DSI64C200K 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 the devices as given in the table below 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.

Setting	Initiator	Target(s)
Initiator Mode	Initiator	Target
Clock Source	External FGEN	External FGEN
External Clock Output Source	Rate Generator	N/A (External FGEN)
SYNC Source	External	External

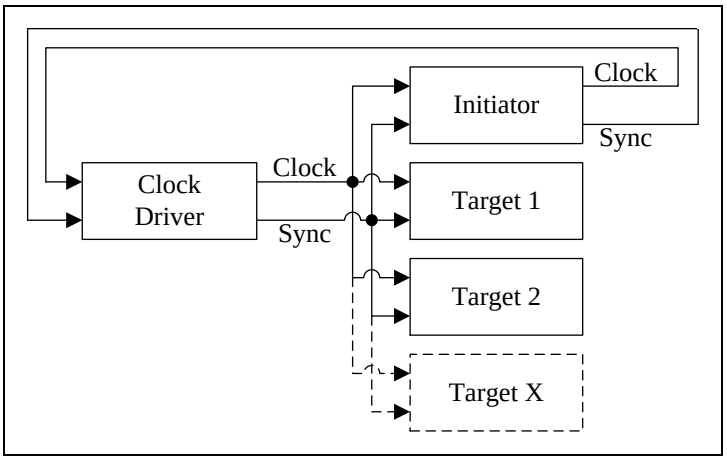

Figure 2 The *star* configuration with three or more boards requires a Clock Driver board.

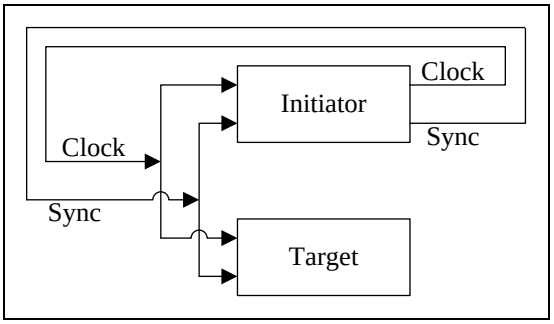

Figure 3 The *star* configuration with only two boards does not require a Clock Driver board.

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
Clock Source	Internal FGEN	External FGEN
External Clock Output Source	Internal FGEN	N/A (External FGEN)
SYNC Source	Internal	External

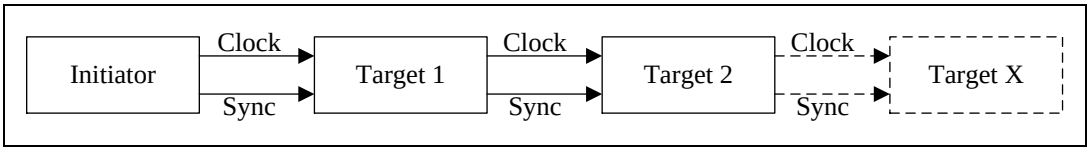

Figure 4 In this configuration the clock and sync signals are daisy chained from one board to the next.

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`". 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. billion - Billion Byte Read - …/billion/

This application configures the designated board then reads in a billion bytes. The data is discarded after it is read.

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 tests of the board 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. 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.

9.9. 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.

9.10. signals - Digital Signals - …/signals/

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

Document History

Revision	Description
April 21, 2023	Updated to version 1.5.103.46.1. Updated the description of the Clear Input Buffer IOCTL service.
March 21, 2023	Updated to version 1.5.103.46.0. Updated the kernel support table.
December 1, 2022	Updated to version 1.4.101.44.0. Updated the kernel support table. Added section on environment variables. Updated the information for the open and close calls. Minor editorial modifications.
October 4, 2021	Updated to release version 1.3.94.37.0. Updated the kernel support table. Minor editorial changes. Added a licensing subsection. Added WAIT_EVENT note. Expanded automatic startup information.
June 20, 2019	Updated to release version 1.2.86.28.0. Updated the kernel support table. Updated the inside cover page. Updated Block Mode DMA macro and associated information. Minor editorial changes. Document reorganization. Added debugging information to the Operating Information section. Some reformatting.
March 19, 2018	Updated to release version 1.1.76.21.0. Updated the CPU and kernel support section. Numerous editorial changes.
May 31, 2017	Updated to release version 1.1.71.20.0. Adjusted the lower sample rate limit.
May 30, 2017	Initial release.