24DSI6LN4AO

6 A/D Channels @ 24-bit, 4 D/A Channels @ 16-bit
With 10-bit Discrete Digital I/O

All Form Factors
…-24DSI6LN4AO

Linux Device Driver And API Library User Manual

Manual Revision: September 28, 2023
Driver Release Version 1.0.105.47.0

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

Preface

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

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

Table of Contents

Table of Figures

1. Introduction

1.1. Purpose

The purpose of this document is to describe the interface to the 24DSI6LN4AO 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 24DSI6LN4AO hardware. The API Library and driver interfaces are based on the board's functionality.

1.2. Acronyms

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

Acronyms	Description
ADC	Analog-to-Digital Converter
API	Application Programming Interface
BMDMA	Block Mode DMA
DAC	Digital-to- Analog Converter
DMA	Direct Memory Access
DMDMA	Demand Mode DMA
GSC	General Standards Corporation
PCI	Peripheral Component Interconnect
PIO	Programmed I/O
PMC	PCI Mezzanine Card
PMC66	This is a PMC formfactor device that can operate at up to 66MHz over the PCI bus.

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 24DSI6LN4AO installation directory or any of its subdirectories.
24DSI6LN4AO	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 24DSI6LN4AO hardware.
Application	This is a user mode process, which runs in user space with user mode privileges.
Driver	This is the 24DSI6LN4AO 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 24DSI6LN4AO applications. The overall architecture is illustrated in Figure 1 below.

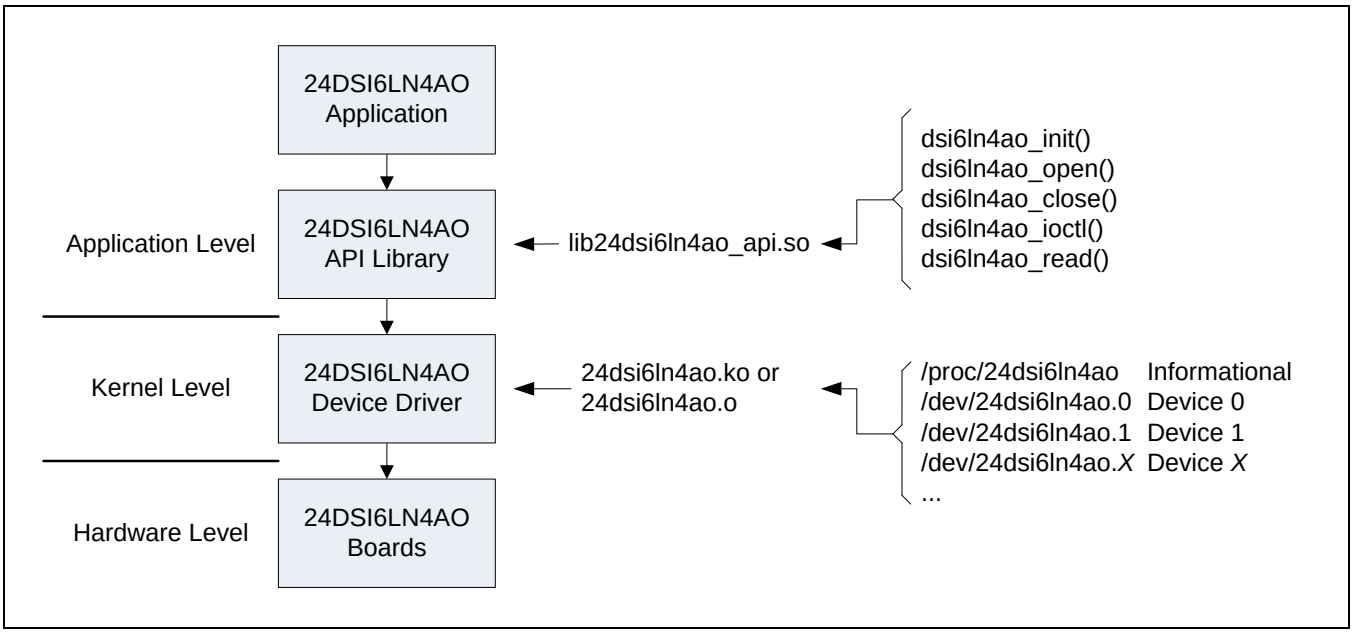

Figure 1 Basic architectural representation.

1.4.2. API Library

The primary means of accessing 24DSI6LN4AO boards is via the 24DSI6LN4AO API Library. This library forms a layer between the application and the driver. Additional information is given in section 4 (page 18). With the library, applications are able to open and close a device and, while open, perform I/O control and read operations.

1.4.3. Device Driver

The device driver is the host software that provides a means of communicating directly with 24DSI6LN4AO 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 24DSI6LN4AO is a high-performance 24-bit analog-to-digital interface board with 16-bit digital-to-analog capabilities. The host side connection is 32-bit PCI based. The external I/O interface may vary per model ordered. The board contains six synchronous 24-bit analog-to-digital input channels capable of performing up to 200,000 conversions per second per channel. All channels are clocked simultaneously. Conversions can be performed on demand or continuously. An onboard receive FIFO of 256k samples collects the converted data for subsequent retrieval by the host. The FIFO allows the 24DSI6LN4AO to buffer data between the cable interface and the PCI bus while maintaining continuous conversions on the cable interface (at least up to the depth of the FIFOs) independent of the PCI bus interface. Converted data can be retrieved using either PIO or DMA. The board also contains four independent, unbuffered 16-bit digital-to-analog output channels that can operate at up to 250,000 conversions per second per channel. In addition, the board includes TTL level digital I/O lines. This consists of an 8-bit bidirectional discrete digital I/O port with one dedicated input and one dedicated output.

1.6. Reference Material

The following reference material may be of particular benefit in using the 24DSI6LN4AO. The specifications provide the information necessary for an in depth understanding of the specialized features implemented on this board.

- The applicable *24DSI6LN4AO User Manual* from General Standards Corporation.
- The *PCI9056 PCI Bus Master Interface Chip* data handbook from PLX Technology, Inc.

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

1.7. Licensing

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

2. Installation

2.1. CPU and Kernel Support

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

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

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

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

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

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

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

2.1.1. 32-bit Support Under 64-bit Environments

This driver supports 32-bit applications under 64-bit environments. The availability of this feature in the kernel depends on a 64-bit kernel being configured to support 32-bit application compatibility. Additionally, 2.6 kernels prior to 2.6.11 implemented 32-bit compatibility in a way that resulted in some drivers not being able to take advantage of the feature. (In these kernels a driver's IOCTL command codes must be globally unique. Beginning with 2.6.11 this requirement has been lifted.) If the driver is not able to provide 32-bit support under a 64-bit kernel, the "`32-bit support`" field in the `/proc/24dsi6ln4ao` file will be "`no`".

2.2. The /proc/ File System

While the driver is running, the text file `/proc/24dsi6ln4ao` can be read to obtain information about the driver and the boards it detects. Each file entry includes an entry name followed immediately by a colon, a space character, and the entry value. Below is an example of what appears in the file, followed by descriptions of each entry.

```
version: 1.0.105.47
32-bit support: yes
boards: 1
models: 24DSI6LN4AO
```

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
`24dsi6ln4ao.linux.tar.gz`	This archive contains the driver, the API Library and all related files.
`24dsi6ln4ao_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
`24dsi6ln4ao/`	This is the driver root directory. It contains the documentation, the Overall Make Script (section 2.7, page 13) and the below listed subdirectories.
`…/api/`	This directory contains the API Library source files (section 4, page 18).
`…/docsrc/`	This directory contains the source files for the code samples given in this document (section 6, page 49).
`…/driver/`	This directory contains the device driver source files (section 5, page 45).
`…/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 50).

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 `24dsi6ln4ao.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 `24dsi6ln4ao` in the current directory, and then copies all of the archive's files into this new directory.

   ```
   tar –xzvf 24dsi6ln4ao.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 48).

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 24dsi6ln4ao.linux.tar.gz 24dsi6ln4ao
   ```

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

   ```
   rm -f /dev/24dsi6ln4ao.*
   ```

5. If the automatic startup procedure was adopted (section 5.3.2, page 46), then edit the system startup script `rc.local` and remove the line that invokes the 24DSI6LN4AO'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 (…`/24dsi6ln4ao/`).

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/lib24dsi6ln4ao_api.so`
Defined and Not Empty	`==== Linking: ../lib/lib24dsi6ln4ao_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/24dsi6ln4ao_utils.a`
Defined and Not Empty	`==== Linking: ../lib/24dsi6ln4ao_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 24DSI6LN4AO 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 24DSI6LN4AO 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 24DSI6LN4AO specific header files. Therefore, sources may include only this one 24DSI6LN4AO header and make files may reference only this one 24DSI6LN4AO include directory.

Description	File	Location
Header File	`24dsi6ln4ao_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 24DSI6LN4AO 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 24DSI6LN4AO static library and only this one 24DSI6LN4AO library directory.

Description	File	Location
Static Library	`24dsi6ln4ao_main.a`	`…/lib/`
	`24dsi6ln4ao_multi.a`	

> **NOTE**: For applications using the 24DSI6LN4AO and no other GSC devices, link the `24dsi6ln4ao_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 24DSI6LN4AO API Library is implemented as a shared library and is thus not linked with the 24DSI6LN4AO Main Library. The API Library must be linked with applications by adding the argument `-l24dsi6ln4ao_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`

4. API Library

The 24DSI6LN4AO API Library is the software interface between user applications and the 24DSI6LN4AO device driver. The interface is accessed by including the header file `24dsi6ln4ao_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	`24dsi6ln4ao_api.h`	`…/include/`
Library File	`lib24dsi6ln4ao_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	`24dsi6ln4ao_api.h`	`…/include/`	
Shared Object Library	`lib24dsi6ln4ao_api.so`	`…/lib/`	
		`/usr/lib/`	`-l24dsi6ln4ao_api`

4.4. Macros

The API Library and driver interfaces include the following macros, which are defined in `24dsi6ln4ao.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 24DSI6LN4AO registers.

4.4.2.1. GSC Registers

The following table gives the complete set of GSC specific 24DSI6LN4AO 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 *24DSI6LN4AO 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
DSI6LN4AO_GSC_BCFGR	Board Configuration Register
DSI6LN4AO_GSC_BCTLR	Board Control Register
DSI6LN4AO_GSC_DIOPR	Digital I/O Port Register
DSI6LN4AO_GSC_IBCR	Input Buffer Control Register
DSI6LN4AO_GSC_IBSR	Input Buffer Size Register
DSI6LN4AO_GSC_IDBR	Input Data Buffer Register
DSI6LN4AO_GSC_IRAR	Input Rate Assignment Register
DSI6LN4AO_GSC_IRDR	Input Rate Divisor Register
DSI6LN4AO_GSC_OC0R	Output Ch 0 Register
DSI6LN4AO_GSC_OC1R	Output Ch 1 Register
DSI6LN4AO_GSC_OC2R	Output Ch 2 Register
DSI6LN4AO_GSC_OC3R	Output Ch 3 Register
DSI6LN4AO_GSC_ORDR	Output Rate Divisor 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 `24dsi6ln4ao_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 `24dsi6ln4ao_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. dsi6ln4ao_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 dsi6ln4ao_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 "24dsi6ln4ao_dsl.h"

int dsi6ln4ao_close_dsl(int fd)
{
    int errs;
    int ret;

    ret = dsi6ln4ao_close(fd);

    if (ret)
        printf("ERROR: dsi6ln4ao_close() returned %d\n", ret);

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

4.6.2. dsi6ln4ao_init()

This function is the entry point to initializing the 24DSI6LN4AO 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 dsi6ln4ao_init(void);
```

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

Example

```
#include <stdio.h>

#include "24dsi6ln4ao_dsl.h"

int dsi6ln4ao_init_dsl(void)
{
    int errs;
    int ret;

    ret = dsi6ln4ao_init();

    if (ret)
        printf("ERROR: dsi6ln4ao_init() returned %d\n", ret);

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

4.6.3. dsi6ln4ao_ioctl()

This function is the entry point to performing setup and control operations on a 24DSI6LN4AO. 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`.

Prototype

```
int dsi6ln4ao_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 "24dsi6ln4ao_dsl.h"

int dsi6ln4ao_ioctl_dsl(int fd, int request, void* arg)
```

```
{
    int errs;
    int ret;

    ret = dsi6ln4ao_ioctl(fd, request, arg);

    if (ret)
        printf("ERROR: dsi6ln4ao_ioctl() returned %d\n", ret);

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

4.6.4. dsi6ln4ao_open()

This function is the entry point to open a connection to a 24DSI6LN4AO board. Before returning, the initialize IOCTL service is called to reset all hardware and software settings to their defaults.

Prototype

```
int dsi6ln4ao_open(int device, int share, int* fd);
```

Argument	Description
device	This is the zero-based index of the 24DSI6LN4AO to access. †
share	Open the device in Shared Access Mode? If non-zero the device is opened in Shared Access Mode (see below). If zero the device is opened in Exclusive Access Mode (see below).
fd	The device handle is returned here. The pointer cannot be NULL. Values returned are as follows.

Value	Description
>= 0	This is the handle to use to access the device in subsequent calls.
-1	There was an error. The device is not accessible.

† 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 "24dsi6ln4ao_dsl.h"

int dsi6ln4ao_open_dsl(int device, int share, int* fd)
{
    int errs;
    int ret;

    ret = dsi6ln4ao_open(device, share, fd);

    if (ret)
        printf("ERROR: dsi6ln4ao_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. dsi6ln4ao_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 8.5, page 52).

Prototype

```
int dsi6ln4ao_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 4.7.41, page 40) before the entire request could be satisfied.
`< 0`	An error occurred. See error value description above.

Example

```
#include <stdio.h>

#include "24dsi6ln4ao_dsl.h"
```

```
int dsi6ln4ao_read_dsl(int fd, void* dst, size_t bytes, size_t* qty)
{
    int errs;
    int ret;

    ret = dsi6ln4ao_read(fd, dst, bytes);

    if (ret < 0)
        printf("ERROR: dsi6ln4ao_read() returned %d\n", ret);

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

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

4.6.6. 24DSI6LN4AO Write

A write function is not supported for generating analog output. Instead, applications must use the `DSI6LN4AO_IOCTL_AO_CH_X_WRITE` IOCTL services (section 4.7.20, page 31).

4.7. IOCTL Services

The 24DSI6LN4AO API Library and device driver implement the following IOCTL services. Each service is described along with the applicable `dsi6ln4ao_ioctl()` function arguments.

4.7.1. DSI6LN4AO_IOCTL_AI_BUF_CLEAR (IBCR D19)

This service clears the data from the input buffer. It also clears the associated overflow and underflow status bits.

Usage

Argument	Description
request	DSI6LN4AO_IOCTL_AI_BUF_CLEAR
arg	Not used.

4.7.2. DSI6LN4AO_IOCTL_AI_BUF_ENABLE (IBCR D18)

This service enables and disables input to the Analog Input buffer.

Usage

Argument	Description
request	DSI6LN4AO_IOCTL_AI_BUF_ENABLE
arg	s32*

Valid argument values are as follows.

Value	Description
-1	This requests the current setting.
DSI6LN4AO_AI_BUF_ENABLE_NO	This disables input to the buffer.
DSI6LN4AO_AI_BUF_ENABLE_YES	This enables input to the buffer.

4.7.3. DSI6LN4AO_IOCTL_AI_BUF_LEVEL (IBSR D0-D18)

This service reports the Input Buffer fill level.

Usage

Argument	Description
request	DSI6LN4AO_IOCTL_AI_BUF_LEVEL
arg	s32*

The values returned by the service are in the inclusive range from zero to 256K (0x40000).

4.7.4. DSI6LN4AO_IOCTL_AI_BUF_OVER (IBCR D24)

This service reports on and clears Input Buffer Overflow status.

Usage

Argument	Description
request	DSI6LN4AO_IOCTL_AI_BUF_OVER
arg	s32*

Valid argument values passed to the service are as follows.

Value	Description
-1	This option requests the current condition.
DSI6LN4AO_AI_BUF_OVER_CLEAR	This clears on Overflow status.
DSI6LN4AO_AI_BUF_OVER_TEST	This option requests the current condition.

Valid argument values returned by the service are as follows.

Value	Description
DSI6LN4AO_AI_BUF_OVER_NO	This indicates than an Overflow has not occurred.
DSI6LN4AO_AI_BUF_OVER_YES	This indicates than an Overflow has occurred.

4.7.5. DSI6LN4AO_IOCTL_AI_BUF_STATUS (BCTLR D23-D24)

This service retrieves the fill level status of the Analog Input buffer.

Usage

Argument	Description
request	DSI6LN4AO_IOCTL_AI_BUF_STATUS
arg	s32*

Valid argument values returned are as follows.

Value	Description
DSI6LN4AO_AI_BUF_STATUS_EMPTY	The buffer is empty.
DSI6LN4AO_AI_BUF_STATUS_PARTIAL	The buffer is neither empty nor full.
DSI6LN4AO_AI_BUF_STATUS_FULL	The buffer is full.

4.7.6. DSI6LN4AO_IOCTL_AI_BUF_UNDER (IBCR D25)

This service reports on and clears Input Buffer Underflow status.

Usage

Argument	Description
request	DSI6LN4AO_IOCTL_AI_BUF_UNDER
arg	s32*

Valid argument values passed to the service are as follows.

Value	Description
-1	This option requests the current condition.
DSI6LN4AO_AI_BUF_UNDER_CLEAR	This clears on Underflow status.
DSI6LN4AO_AI_BUF_UNDER_TEST	This option requests the current condition.

Valid argument values returned by the service are as follows.

Value	Description
DSI6LN4AO_AI_BUF_UNDER_NO	This indicates than an Underflow has not occurred.
DSI6LN4AO_AI_BUF_UNDER_YES	This indicates than an Underflow has occurred.

4.7.7. DSI6LN4AO_IOCTL_AI_CHAN_LAST (BCTLR D16-D18)

This service configures the selection of the last channel to scan, which effectively sets the range and number of channels to scan.

Usage

Argument	Description
request	DSI6LN4AO_IOCTL_AI_CHAN_LAST
arg	s32*

Valid argument values are as follows.

Value	Description
-1	Retrieve the current setting.
0	Scan channel 0 only.
1	Scan channels 0-1.
2	Scan channels 0-2.
3	Scan channels 0-3.
4	Scan channels 0-4.
5	Scan channels 0-5.
6	Scan channels 0-6.
7	Scan channels 0-7.

4.7.8. DSI6LN4AO_IOCTL_AI_COUPLING (BCTLR D22)

This service configures the Analog Input Coupling Mode.

Usage

Argument	Description
request	DSI6LN4AO_IOCTL_AI_COUPLING
arg	s32*

Valid argument values are as follows.

Value	Description
-1	Passing in this value requests the current setting.
DSI6LN4AO_AI_COUPLING_AC	This is AC coupling, which is for most operational conditions.
DSI6LN4AO_AI_COUPLING_DC	This is DC coupling, which is for testing only.

4.7.9. DSI6LN4AO_IOCTL_AI_DATA_WIDTH (IBCR D20-D21)

This service configures the Analog Input Data Width.

Usage

Argument	Description
request	DSI6LN4AO_IOCTL_AI_DATA_WIDTH
arg	s32*

Valid argument values are as follows.

Value	Description
-1	Passing in this value requests the current setting.
DSI6LN4AO_AI_DATA_WIDTH_16	This selects 16-bit results.
DSI6LN4AO_AI_DATA_WIDTH_18	This selects 18-bit results.
DSI6LN4AO_AI_DATA_WIDTH_20	This selects 20-bit results.
DSI6LN4AO_AI_DATA_WIDTH_24	This selects 24-bit results.

4.7.10. DSI6LN4AO_IOCTL_AI_MODE (BCTLR D0)

This service configures the Analog Input Mode.

Usage

Argument	Description
request	DSI6LN4AO_IOCTL_AI_MODE
arg	s32*

Valid argument values are as follows.

Value	Description
-1	Passing in this value requests the current setting.
DSI6LN4AO_AI_MODE_DIFF	This selects Differential input processing.
DSI6LN4AO_AI_MODE_SE	This selects Single Ended input processing.

4.7.11. DSI6LN4AO_IOCTL_AI_NDIV (IRDR D0-D5)

This service sets the NDIV value for input sample rate generation.

Usage

Argument	Description
request	DSI6LN4AO_IOCTL_AI_NDIV
arg	s32*

Valid argument values are as follows.

Value	Description
-1	Passing in this value requests the current setting.
0-25	This is the inclusive range of valid values.

4.7.12. DSI6LN4AO_IOCTL_AI_NREF (IRAR D16-D25)

This service sets the NREF value for input sample rate generation.

Usage

Argument	Description
request	DSI6LN4AO_IOCTL_AI_NREF
arg	s32*

Valid argument values are as follows.

Value	Description
-1	Passing in this value requests the current setting.
30-1000	This is the inclusive range of valid values.

4.7.13. DSI6LN4AO_IOCTL_AI_NVCO (IRAR D0-D9)

This service sets the NVCO value for input sample rate generation.

Usage

Argument	Description
request	DSI6LN4AO_IOCTL_AI_NVCO
arg	s32*

Valid argument values are as follows.

Value	Description
-1	Passing in this value requests the current setting.
30-1000	This is the inclusive range of valid values.

4.7.14. DSI6LN4AO_IOCTL_AI_READY (BCTLR D20)

This service deals with the Analog Input Ready status.

Usage

Argument	Description
request	DSI6LN4AO_IOCTL_AI_READY
arg	s32*

Valid argument values passed to the service are as follows.

Value	Description
-1	This option requests the current condition.
DSI6LN4AO_AI_READY_TEST	This option requests the current condition.
DSI6LN4AO_AI_READY_WAIT	This option tells the driver to wait until the status is asserted before returning from the request. The driver waits for up to 500 milliseconds for the status to be asserted. If the status is not asserted in that time period, then the service returns the timeout error status.

Valid argument values returned by the service are as follows.

Value	Description
DSI6LN4AO_AI_READY_NO	This indicates than the status is not asserted.
DSI6LN4AO_AI_READY_YES	This indicates than the status is asserted.

4.7.15. DSI6LN4AO_IOCTL_AI_SAMPLING (BCTLR D1)

This service sets the Analog Input Sampling mode.

Usage

Argument	Description
request	DSI6LN4AO_IOCTL_AI_SAMPLING
arg	s32*

Valid argument values are as follows.

Value	Description
-1	Passing in this value requests the current setting.
DSI6LN4AO_AI_SAMPLING_BURST	Input acquisition is initiated by an AI Trigger request.
DSI6LN4AO_AI_SAMPLING_CONT	Input acquisition is initiated by the internal clock.

4.7.16. DSI6LN4AO_IOCTL_AI_SYNC (BCTLR D7)

This service initiates an input synchronization operation, which is required before the input channels perform acquisition in unison. If synchronization is not performed, then channel data may appear out of phase with one another even though acquisition is performed at the same rate.

Usage

Argument	Description
request	DSI6LN4AO_IOCTL_AI_SYNC
arg	Not used.

4.7.17. DSI6LN4AO_IOCTL_AI_THRESH (IBCR D0-D17)

This service sets the Analog Input Buffer fill level threshold above which the Analog Input Threshold Status is asserted.

Usage

Argument	Description
request	DSI6LN4AO_IOCTL_AI_THRESH
arg	s32*

Valid argument values are as follows.

Value	Description
-1	Passing in this value requests the current setting.
0-0x3FFFF	This is the inclusive range of valid values.

4.7.18. DSI6LN4AO_IOCTL_AI_THRESH_STS (BCTLR D25)

This service reports the Analog Input Buffer fill level relative to the programmed threshold value. If the Analog Input Buffer fill level exceeds the value of the Analog Input Buffer Threshold, then the status is asserted. The status is otherwise negated.

Usage

Argument	Description
request	DSI6LN4AO_IOCTL_AI_THRESH_STS
arg	s32*

Valid argument values are as follows.

Value	Description
-1	Passing in this value requests the current setting.
DSI6LN4AO_AI_THRESH_STS_CLEAR	The input buffer fill level is at or below the threshold.
DSI6LN4AO_AI_THRESH_STS_SET	The input buffer fill level is above below the threshold.

4.7.19. DSI6LN4AO_IOCTL_AI_TRIGGER (BCTLR D21)

This service initiates an input acquisition of all active channels when the acquisition mode is configured for burst operation.

Usage

Argument	Description
request	DSI6LN4AO_IOCTL_AI_TRIGGER
arg	Not used.

4.7.20. DSI6LN4AO_IOCTL_AO_CH_X_WRITE (OC0R-OC3R)

This refers to the below listed services.

Service	Description
DSI6LN4AO_IOCTL_AO_CH_0_WRITE	Write to Output Channel 0.
DSI6LN4AO_IOCTL_AO_CH_1_WRITE	Write to Output Channel 1.
DSI6LN4AO_IOCTL_AO_CH_2_WRITE	Write to Output Channel 2.
DSI6LN4AO_IOCTL_AO_CH_3_WRITE	Write to Output Channel 3.

These services write a value to their respective Analog Output channels.

Usage

Argument	Description
request	DSI6LN4AO_IOCTL_AO_CH_*X*_WRITE
arg	s32*

Valid argument values are as follows.

Value	Description
-1	Passing in this value requests the current setting.
0-0xFFFF	This is the value to be applied to the Output Channel.

4.7.21. DSI6LN4AO_IOCTL_AO_CLK_MODE (BCTLR D29)

This service configures the output Clock Mode.

Usage

Argument	Description
request	DSI6LN4AO_IOCTL_AO_CLK_MODE
arg	s32*

Valid argument values are as follows.

Value	Description
-1	Passing in this value requests the current setting.
DSI6LN4AO_AO_CLK_MODE_ASYNC	This selects Asynchronous operation in which DAC values are posted to the output immediately.
DSI6LN4AO_AO_CLK_MODE_SYNC	This selects Synchronous operation in which DAC values are posted to the output only when clocked.

4.7.22. DSI6LN4AO_IOCTL_AO_CLK_SRC (BCTLR D26)

This service selects the source that clocks Analog Output when operating in Synchronous Clock Mode.

Usage

Argument	Description
request	DSI6LN4AO_IOCTL_AO_CLK_SRC
arg	s32*

Valid argument values are as follows.

Value	Description
-1	Passing in this value requests the current setting.
DSI6LN4AO_AO_CLK_SRC_EXT	This selects the AO clock input at the cable interface.
DSI6LN4AO_AO_CLK_SRC_INT	This selects the internal output rate generator.

4.7.23. DSI6LN4AO_IOCTL_AO_ENABLE (BCTLR D30)

This service enables or disables the analog outputs.

Usage

Argument	Description
request	DSI6LN4AO_IOCTL_AO_ENABLE
arg	s32*

Valid argument values are as follows.

Value	Description
-1	Passing in this value requests the current setting.
DSI6LN4AO_AO_ENABLE_NO	This disables Analog Output by disconnecting the signals from the cable interface.
DSI6LN4AO_AO_ENABLE_YES	This enables Analog Output by connecting the signals to the cable interface.

4.7.24. DSI6LN4AO_IOCTL_AO_NRATE (ORDR D0-D23)

This service sets the NRATE value for output clock rate generation.

Usage

Argument	Description
request	DSI6LN4AO_IOCTL_AO_NRATE
arg	s32*

Valid argument values are as follows.

Value	Description
-1	Passing in this value requests the current setting.
0-0xFFFFFF	This is the inclusive range of valid values.

4.7.25. DSI6LN4AO_IOCTL_AO_RANGE (BCTLR D2-D3)

This service selects the Analog Output voltage range.

Usage

Argument	Description
request	DSI6LN4AO_IOCTL_AO_RANGE
arg	s32*

Valid argument values are as follows.

Value	Description
-1	Passing in this value requests the current setting.
DSI6LN4AO_AO_RANGE_2_5V	This selects the ±2.5 Volt range.
DSI6LN4AO_AO_RANGE_5V	This selects the ±5 Volt range.
DSI6LN4AO_AO_RANGE_10V	This selects the ±10 Volt range.

4.7.26. DSI6LN4AO_IOCTL_AO_READY (BCTLR D27)

This service deals with the Analog Input Ready status.

Usage

Argument	Description
request	DSI6LN4AO_IOCTL_AO_READY
arg	s32*

Valid argument values passed to the service are as follows.

Value	Description
-1	This option requests the current condition.
DSI6LN4AO_AO_READY_TEST	This option requests the current condition.
DSI6LN4AO_AO_READY_WAIT	This option tells the driver to wait until the status is asserted before returning from the request. The driver waits for 500 milliseconds for the status to be asserted. If the status is not asserted in that time period, then the service returns the timeout error status.

Valid argument values returned by the service are as follows.

Value	Description
DSI6LN4AO_AO_READY_NO	This indicates than the status is not asserted.
DSI6LN4AO_AO_READY_YES	This indicates than the status is asserted.

4.7.27. DSI6LN4AO_IOCTL_AO_TRIGGER (BCTLR D28)

This service clocks DAC values to the outputs if the AO Clock Mode is set to Synchronous. This option clocks output even though output clocking is also being driven by the internal or external output clock.

Usage

Argument	Description
request	DSI6LN4AO_IOCTL_AO_TRIGGER
arg	Not used.

4.7.28. DSI6LN4AO_IOCTL_DATA_FORMAT (BCTLR D5)

This service sets the Analog Input and Output data encoding format.

Usage

Argument	Description
request	DSI6LN4AO_IOCTL_DATA_FORMAT
arg	s32*

Valid argument values returned are as follows.

Value	Description
-1	This requests the current setting.
DSI6LN4AO_DATA_FORMAT_2S_COMP	This refers to the Twos Compliment encoding format.
DSI6LN4AO_DATA_FORMAT_OFF_BIN	This refers to the Offset Binary encoding format.

4.7.29. DSI6LN4AO_IOCTL_GPIO_DIR (DIOPR D0-D7, D9-D10)

This service sets the direction of the 8-bit digital I/O port as well as the signal states for those signals operating as outputs.

Usage

Argument	Description
request	DSI6LN4AO_IOCTL_GPIO_DIR
arg	s32*

Valid argument values controlling the port direction are as follows. If the bits for the outputs are left clear, then the output port pins are driven low. Any output bits that are set will cause the port pins to be driven high.

Value	Description
-1	This requests the current setting.
DSI6LN4AO_GPIO_DIR_INPUT	This selects the input direction.
DSI6LN4AO_GPIO_DIR_OUTPUT	This selects the output direction.

4.7.30. DSI6LN4AO_IOCTL_GPIO_RX (DIOPR D0-D9)

This service reads the value at all 10 port pins.

Usage

Argument	Description
request	DSI6LN4AO_IOCTL_GPIO_RX
arg	s32*

Valid argument values returned are in the inclusive range from zero to 0x3FF.

4.7.31. DSI6LN4AO_IOCTL_GPIO_TX (DIOPR D0-D7, D9)

This service sets the output level for the output port pins. If the bidirectional pins are configured as inputs, then those bits have no effect. Port pin D8, which corresponds to the dedicated input, is ignored.

Usage

Argument	Description
request	DSI6LN4AO_IOCTL_GPIO_TX
arg	s32*

Valid argument values are in the inclusive range from zero through 0x3FF.

4.7.32. DSI6LN4AO_IOCTL_INITIALIZE (BCTLR D31)

This service returns all registers to their initialized state and initializes all API software settings.

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

Usage

Argument	Description
request	DSI6LN4AO_IOCTL_INITIALIZE
arg	Not used.

4.7.33. DSI6LN4AO_IOCTL_IRQ_SEL (BCTLR D8-D10, D11)

This service configures the source selection for firmware interrupts.

Usage

Argument	Description
request	DSI6LN4AO_IOCTL_IRQ_SEL
arg	s32*

Valid argument values are as follows.

Value	Description
-1	Retrieve the current setting.
DSI6LN4AO_IRQ_AI_BUF_ERROR	This refers to an Input Buffer Overflow or Underflow.
DSI6LN4AO_IRQ_AI_BUF_THR_H2L	This refers to the Input Buffer Threshold Status going low.
DSI6LN4AO_IRQ_AI_BUF_THR_L2H	This refers to the Input Buffer Threshold Status going high.
DSI6LN4AO_IRQ_AI_READY_L2H	This refers to the AI Ready status going high.
DSI6LN4AO_IRQ_AO_CLOCK_DONE	This refers to the completion of an output clocking operation.
DSI6LN4AO_IRQ_INIT_DONE	This refers to the completion of an initialization cycle.

4.7.34. DSI6LN4AO_IOCTL_QUERY

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

Usage

Argument	Description
request	DSI6LN4AO_IOCTL_QUERY
arg	s32*

Valid argument values are as follows.

Value	Description
DSI6LN4AO_QUERY_AI_CHAN_MAX	This is the board's maximum number of input channels.
DSI6LN4AO_QUERY_AI_CHAN_QTY	This is the number of input channels installed on the board.
DSI6LN4AO_QUERY_AI_FIFO_SIZE	This is the size of the input buffer in 32-bit A/D values.
DSI6LN4AO_QUERY_AI_FILTER	This is the input filter option as reported by BCFGR D20.
DSI6LN4AO_QUERY_AI_FREF	This is the AI master clock frequency in Hz.
DSI6LN4AO_QUERY_AI_FSAMP_MAX	This gives the maximum AI FSAMP value in S/S.
DSI6LN4AO_QUERY_AI_FSAMP_MIN	This gives the minimum AI FSAMP value in S/S.
DSI6LN4AO_QUERY_AI_NDIV_MAX	This is the maximum AI NDIV rate generator value.
DSI6LN4AO_QUERY_AI_NDIV_MIN	This is the minimum AI NDIV rate generator value.
DSI6LN4AO_QUERY_AI_NREF_MAX	This is the maximum overall AI NREF rate generator value.
DSI6LN4AO_QUERY_AI_NREF_MAX_OPT	This is the maximum optimal AI NREF rate generator value.
DSI6LN4AO_QUERY_AI_NREF_MIN	This is the minimum AI NREF rate generator value.
DSI6LN4AO_QUERY_AI_NVCO_MAX	This is the maximum overall AI NVCO rate generator value.
DSI6LN4AO_QUERY_AI_NVCO_MAX_OPT	This is the maximum optimal AI NVCO rate generator value.
DSI6LN4AO_QUERY_AI_NVCO_MIN	This is the minimum AI NVCO rate generator value.
DSI6LN4AO_QUERY_AI_RANGE	This is the board's supported input voltage range as reported by BCFGR D18-D19.
DSI6LN4AO_QUERY_AO_CHAN_MAX	This is the board's maximum number of output channels.
DSI6LN4AO_QUERY_AO_CHAN_QTY	This is the number of output channels installed on the board.
DSI6LN4AO_QUERY_AO_FCLK_MAX	This gives the maximum AO FCLK value in Hz.
DSI6LN4AO_QUERY_AO_FCLK_MIN	This gives the minimum AO FCLK value in Hz.
DSI6LN4AO_QUERY_AO_FREF	This is the AO master clock frequency in Hz.
DSI6LN4AO_QUERY_AO_NRATE_MAX	This is the maximum AO NRATE rate generator value.
DSI6LN4AO_QUERY_AO_NRATE_MIN	This is the minimum AO NRATE rate generator value.
DSI6LN4AO_QUERY_COUNT	This returns the number of query options supported by the IOCTL service.
DSI6LN4AO_QUERY_DEVICE_TYPE	This returns the identifier value for the board's type. This should be GSC_DEV_TYPE_24DSI6LN4AO.
DSI6LN4AO_QUERY_INIT_MS	This returns the duration of a board initialization in milliseconds.
DSI6LN4AO_QUERY_TEMPERATURE	This is the board's temperature option as reported by BCFGR D21.

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

Value	Description
DSI6LN4AO_IOCTL_QUERY_ERROR	Either there was a processing error or the query option is unrecognized.

Valid return values for the DSI6LN4AO_QUERY_AI_FILTER option are as follows.

Value	Description
DSI6LN4AO_AI_FILTER_270KHZ	The board's input filter is 270KHz.
DSI6LN4AO_AI_FILTER_CUSTOM	The board has a custom input filter.

Valid return values for the DSI6LN4AO_QUERY_AI_RANGE option are as follows.

Value	Description
DSI6LN4AO_AI_RANGE_2_5	The board is configured for ±2.5 volts.
DSI6LN4AO_AI_RANGE_5	The board is configured for ±5V volts.
DSI6LN4AO_AI_RANGE_10	The board is configured for ±10V volts.

Valid return values for the DSI6LN4AO_QUERY_TEMPERATURE option are as follows.

Value	Description
DSI6LN4AO_TEMPERATURE_COM	The board supports the commercial temperature range.
DSI6LN4AO_TEMPERATURE_EXT	The board supports the extended temperature range.

4.7.35. DSI6LN4AO_IOCTL_REG_MOD

This service performs a read-modify-write operation on a board register. This includes only the firmware registers, as the PCI and PLX Feature Set registers are read-only.

Usage

Argument	Description
request	DSI6LN4AO_IOCTL_REG_MOD
arg	gsc_reg_t*

Definition

```
typedef struct
{
    u32 reg;    // range: any valid register definition
    u32 value;  // range: 0x0-0xFFFFFFFF
    u32 mask;   // range: 0x0-0xFFFFFFFF
} gsc_reg_t;
```

Fields	Description
reg	This is the register to access. Refer to section 4.4.2 on page 19 for additional information.
value	This is the value to write to the specified register. Only the bits set in the map are applied.
mask	This is a map of the bits to modify. If a bit is set, then the corresponding register bit is set according the content of the value field. If a bit here is zero, then that register bit is unmodified.

4.7.36. DSI6LN4AO_IOCTL_REG_READ

This service reads the value of a 24DSI6LN4AO register. This includes the PCI registers, the PLX Feature Set registers and the firmware registers.

Usage

Argument	Description
request	DSI6LN4AO_IOCTL_REG_READ
arg	gsc_reg_t*

Definition

```
typedef struct
{
    u32 reg;    // range: any valid register definition
    u32 value;  // range: 0x0-0xFFFFFFFF
    u32 mask;   // range: 0x0-0xFFFFFFFF
} gsc_reg_t;
```

Fields	Description
reg	This is the register to read from. Refer to section 4.4.2 on page 19 for additional information.
value	This is the value read from the specified register.
mask	This is ignored for read requests.

4.7.37. DSI6LN4AO_IOCTL_REG_WRITE

This service writes a value to a board register. This includes only the firmware registers, as the PCI and PLX Feature Set registers cannot be modified.

Usage

Argument	Description
request	DSI6LN4AO_IOCTL_REG_WRITE
arg	gsc_reg_t*

Definition

```
typedef struct
{
    u32 reg;    // range: any valid register definition
    u32 value;  // range: 0x0-0xFFFFFFFF
    u32 mask;   // range: 0x0-0xFFFFFFFF
} gsc_reg_t;
```

Fields	Description
reg	This is the register to write to. Refer to section 4.4.2 on page 19 for additional information.

value	This is the value to write to the specified register.
mask	This is ignored for write requests.

4.7.38. DSI6LN4AO_IOCTL_RX_IO_ABORT

This service aborts an ongoing dsi6ln4ao_read() request.

Usage

Argument	Description
request	DSI6LN4AO_IOCTL_RX_IO_ABORT
arg	s32*

The results are reported as one of the following values.

Value	Description
DSI6LN4AO_IO_ABORT_NO	A read request was not aborted as none were ongoing.
DSI6LN4AO_IO_ABORT_YES	An ongoing read request was aborted.

4.7.39. DSI6LN4AO_IOCTL_RX_IO_MODE

This service selects the data transfer mode for I/O operations. Refer to the dsi6ln4ao_read() service for additional information (section 4.6.5 on page 23).

Usage

Argument	Description
request	DSI6LN4AO_IOCTL_RX_IO_MODE
arg	s32*

Valid argument values are as follows.

Value	Description
-1	This requests the current setting.
GSC_IO_MODE_BMDMA	This refers to Block Mode DMA in which the DMA is initiated only after the data becomes available.
GSC_IO_MODE_DMDMA	This refers to Demand Mode DMA in which the transfer occurs as the data become available. This is the most efficient option for most I/O requests.
GSC_IO_MODE_PIO	This refers to PIO in which data is transferred by repetitive register accesses. This is preferred for very small transfer requests. This is the default.

4.7.40. DSI6LN4AO_IOCTL_RX_IO_OVERFLOW

This service configures the API's response to AI Buffer Overflows.

Usage

Argument	Description
request	DSI6LN4AO_IOCTL_RX_IO_OVERFLOW
arg	s32*

Valid argument values are as follows.

Value	Description
-1	This requests the current setting.
DSI6LN4AO_IO_OVERFLOW_CHECK	The read service checks for AI Buffer Overflows and returns an error when an overflow occurs.
DSI6LN4AO_IO_OVERFLOW_IGNORE	The read service does not check for AI Buffer Overflows.

4.7.41. DSI6LN4AO_IOCTL_RX_IO_TIMEOUT

This service sets the timeout limit for I/O requests. The limit is specified in seconds.

Usage

Argument	Description
request	DSI6LN4AO_IOCTL_RX_IO_TIMEOUT
arg	s32*

Valid argument values are as follows.

Value	Description
-1	This requests the current setting.
0	Do not sleep to wait for data.
1-3600	As needed, wait at most the specified number of seconds for data before returning to the caller.
DSI6LN4AO_IO_TIMEOUT_INFINITE	Do not return until the read request is satisfied.

4.7.42. DSI6LN4AO_IOCTL_RX_IO_UNDERFLOW

This service configures the API's response to AI Buffer Underflows.

Usage

Argument	Description
request	DSI6LN4AO_IOCTL_RX_IO_UNDERFLOW
arg	s32*

Valid argument values are as follows.

Value	Description
-1	This requests the current setting.
DSI6LN4AO_IO_UNDERFLOW_CHECK	The read service checks for AI Buffer Underflows and returns an error when an underflow occurs.
DSI6LN4AO_IO_UNDERFLOW_IGNORE	The read service does not check for AI Buffer Underflows.

4.7.43. DSI6LN4AO_IOCTL_WAIT_CANCEL

This service resumes all threads blocked via DSI6LN4AO_IOCTL_WAIT_EVENT IOCTL calls (section 4.7.44, page 41), 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	DSI6LN4AO_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 4.7.44.2 on page 42.
gsc	This specifies the set of DSI6LN4AO_WAIT_GSC_* events whose wait requests are to be cancelled. Refer to section 4.7.44.3 on page 42.
alt	This is unused with the 24DSI6LN4AO board and should be zero.
io	This specifies the set of DSI6LN4AO_WAIT_IO_* events whose wait requests are to be cancelled. Refer to section 4.7.44.4 on page 43.
timeout_ms	This is unused by wait cancel operations.
count	Upon return this indicates the number of waits that were cancelled.

4.7.44. DSI6LN4AO_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	DSI6LN4AO_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 4.7.44.1on page 42.
main	This specifies any number of GSC_WAIT_MAIN_* events that the thread is to wait for. Refer to section 4.7.44.2 on page 42.
gsc	This specifies any number of DSI6LN4AO_WAIT_GSC_* events that the thread is to wait for. Refer to section 4.7.44.3 on page 42.
alt	This is unused with the 24DSI6LN4AO board and must be zero.
io	This specifies any number of DSI6LN4AO_WAIT_IO_* events that the thread is to wait for. Refer to section 4.7.44.4 on page 43.
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.

4.7.44.1. 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.

4.7.44.2. 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 24DSI6LN4AO 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 24DSI6LN4AO.
GSC_WAIT_MAIN_PCI	This refers to any interrupt generated by the 24DSI6LN4AO.
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.

4.7.44.3. 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 interrupts (section 4.7.33, page 35).

Value	Description
DSI6LN4AO_WAIT_GSC_AI_BUF_ERROR	This refers to an Input Buffer Overflow or Underflow.
DSI6LN4AO_WAIT_GSC_AI_BUF_THR_H2L	This refers to the AI Buffer Threshold Status going low.
DSI6LN4AO_WAIT_GSC_AI_BUF_THR_L2H	This refers to the AI Buffer Threshold Status going high.
DSI6LN4AO_WAIT_GSC_AI_READY_L2H	This refers to the AI Ready status going high.
DSI6LN4AO_WAIT_GSC_AO_CLOCK_DONE	This refers to the completion of an output clocking operation.
DSI6LN4AO_WAIT_GSC_INIT_DONE	This refers to the completion of an initialization cycle.

4.7.44.4. gsc_wait_t.io Options

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

Fields	Description
DSI6LN4AO_WAIT_IO_RX_ABORT	This refers to read requests which have been aborted.
DSI6LN4AO_WAIT_IO_RX_DONE	This refers to read requests which have been satisfied.
DSI6LN4AO_WAIT_IO_RX_ERROR	This refers to read requests which end due to an error.
DSI6LN4AO_WAIT_IO_RX_TIMEOUT	This refers to read requests which end due to the timeout period lapse.

4.7.45. DSI6LN4AO_IOCTL_WAIT_STATUS

This service counts all threads blocked via the DSI6LN4AO_IOCTL_WAIT_EVENT IOCTL service (section 4.7.44, page 41), 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	DSI6LN4AO_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 4.7.44.2 on page 42.
gsc	This specifies the set of DSI6LN4AO_WAIT_GSC_* events whose wait requests are to be counted. Refer to section 4.7.44.3 on page 42.

`alt`	This is unused with the 24DSI6LN4AO board and should be zero.
`io`	This specifies the set of `DSI6LN4AO_WAIT_IO_*` events whose wait requests are to be counted. Refer to section 4.7.44.4 on page 43.
`timeout_ms`	This is unused by wait status operations.
`count`	Upon return this indicates the number of waits that met any of the specified criteria.

5. The Driver

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

5.1. Files

The device driver files are summarized in the table below.

Description	Files	Location
Source Files	`*.c`, `*.h` …	…`/driver/`
Header File	`24dsi6ln4ao.h`	
Driver File	`24dsi6ln4ao.ko` † `24dsi6ln4ao.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 24DSI6LN4AO 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 `24dsi6ln4ao` 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/24dsi6ln4ao.*
   ```

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/24dsi6ln4ao/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/24dsi6ln4ao` 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/24dsi6ln4ao
   ```

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/24dsi6ln4ao` 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 24dsi6ln4ao
   ```

2. Verify that the driver module has been unloaded by issuing the below command. The module name `24dsi6ln4ao` 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	`24dsi6ln4ao_dsl.h`	`…/include/`
Library File	`24dsi6ln4ao_dsl.a`	`…/lib/`

6.2. Build

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

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

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

   ```
   make clean
   ```

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

   ```
   make
   ```

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

6.3. Library Use

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

7. Utilities Source Code

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

For each API function there is a corresponding utility source file with a corresponding utility service. As an example, for the API function `dsi6ln4ao_open()` there is the utility file `open.c` containing the utility function `dsi6ln4ao_open_util()`. The naming pattern is as follows: API function `dsi6ln4ao_xxxx()`, utility file name `xxxx.c`, utility function `dsi6ln4ao_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 `DSI6LN4AO_IOCTL_QUERY` there is the utility file `query.c` containing the utility function `dsi6ln4ao_query()`. The naming pattern is as follows: IOCTL code `DSI6LN4AO_IOCTL_XXXX`, utility file name `xxxx.c`, utility function `dsi6ln4ao_xxxx()`.

7.1. Files

The utility files are summarized in the table below.

Description	Files	Location
Source Files	`*.c, *.h …`	`…/utils/`
Header File	`24dsi6ln4ao_utils.h`	`…/include/`

Library Files	`24dsi6ln4ao_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 24DSI6LN4AO. 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	`dsi6ln4ao_reg_list()`	Source File
Source File	`reg.c`	`…/utils/`
Header File	`24dsi6ln4ao_utils.h`	`…/include/`
Library File	`24dsi6ln4ao_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	`dsi6ln4ao_config_ai()`	Source File
Source File	`config_ai.c`	`…/utils/`
Header File	`24dsi6ln4ao_utils.h`	`…/include/`
Library File	`24dsi6ln4ao_utils.a`	`…/lib/`

8.3. Analog Output Configuration

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

Item	Name/File	Location
Function	dsi6ln4ao_config_ao()	Source File
Source File	config_ao.c	…/utils/
Header File	24dsi6ln4ao_utils.h	…/include/
Library File	24dsi6ln4ao_utils.a	…/lib/

8.4. Digital Input/Output Configuration

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

Item	Name/File	Location
Function	24dsi6ln4ao_config_dio()	Source File
Source File	config_dio.c	…/utils/
Header File	24dsi6ln4ao_utils.h	…/include/
Library File	24dsi6ln4ao_utils.a	…/lib/

8.5. Data Transfer Modes

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

8.5.1. PIO - 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.

8.5.2. 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.

8.5.3. 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.

9. Sample Applications

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

9.1. aout - Analog Output - …/aout/

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

9.2. din - Digital Input - …/din/

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

9.3. dout - Digital Output - …/dout/

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

9.4. fclk – Analog Output Clock Rate - …/fclk/

This application computes and reports the Analog Output clocking configuration most suited to producing a user specified clock rate.

9.5. fsamp – Analog Input Sample Rate - …/fsamp/

This application computes and reports the Analog Input clocking configuration most suited to producing a user specified sample rate.

id - Identify Board - …/id/

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

9.6. regs - Register Access - …/regs/

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

9.7. rxrate - Receive Rate - …/rxrate/

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

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

This application configures the board for a modest sample rate, reads a million samples of data, then saves the data to disk.

9.9. stream – Stream Performance Testing - …/stream/

This application performs multithreaded data streaming by way of Rx and a Tx data threads in order to characterize the sustainable data transfer capabilities of the device and the host. The Rx and Tx threads operations are configurable so that multiple configurations can be tested.

9.10. wait – Wait Event Tests - …/wait/

This application tests the Wait Event services to verify the functionality of all supported wait options, including all supported interrupt options.

Document History

Revision	Description
September 28, 2023	Initial release, version 1.0.105.47.0.