

24DSI

24-bit, 4 to 32 channel, 200KS/S/Ch Delta-Sigma A/D Input

**All Form Factors
...-24DSI32/12/8/6**

Linux Device Driver And API Library User Manual

**Manual Revision: March 24, 2023
Driver Release Version 4.17.107.50.0**

**General Standards Corporation
8302A Whitesburg Drive
Huntsville, AL 35802
Phone: (256) 880-8787
Fax: (256) 880-8788**

URL: <http://www.generalstandards.com>

E-mail: sales@generalstandards.com

E-mail: support@generalstandards.com

Preface

Copyright © 2008-2023, **General Standards Corporation**

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

General Standards Corporation makes no warranty of any kind with regard to this documentation and/or software, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. Although extensive editing and reviews are performed before release, **General Standards Corporation** assumes no responsibility for any errors, inaccuracies or omissions herein. This documentation, information and software are made available solely on an “as-is” basis. Nor is there any commitment to update or keep current this documentation.

General Standards Corporation does not assume any liability arising out of the application or use of documentation, software, product or circuit described herein, nor is any license conveyed under any patent rights or any rights of others.

General Standards Corporation assumes no responsibility for any consequences resulting from omissions or errors in this manual or from the use of information contained herein.

General Standards Corporation reserves the right to make any changes, without notice, to this documentation, software or product, to improve accuracy, clarity, reliability, performance, function, or design.

ALL RIGHTS RESERVED.

GSC is a trademark of **General Standards Corporation**.

PLX and PLX Technology are trademarks of PLX Technology, Inc.

Table of Contents

1. Introduction.....	8
1.1. Purpose.....	8
1.2. Acronyms.....	8
1.3. Definitions	8
1.4. Software Overview	8
1.4.1. Basic Software Architecture	8
1.4.2. API Library.....	9
1.4.3. Device Driver	9
1.5. Hardware Overview	9
1.6. Reference Material.....	9
1.7. Licensing.....	10
2. Installation	11
2.1. CPU and Kernel Support.....	11
2.1.1. 32-bit Support Under 64-bit Environments	12
2.2. The /proc/ File System	12
2.3. File List.....	12
2.4. Directory Structure.....	12
2.5. Installation	13
2.6. Removal.....	13
2.7. Overall Make Script.....	13
2.8. Environment Variables	14
2.8.1. GSC_API_COMP_FLAGS.....	14
2.8.2. GSC_API_LINK_FLAGS.....	14
2.8.3. GSC_LIB_COMP_FLAGS.....	14
2.8.4. GSC_LIB_LINK_FLAGS.....	15
2.8.5. GSC_APP_COMP_FLAGS.....	15
2.8.6. GSC_APP_LINK_FLAGS.....	15
3. Main Interface Files.....	16
3.1. Main Header File	16
3.2. Main Library File.....	16
3.2.1. Build	16
3.2.2. System Libraries.....	17
4. API Library	18
4.1. Files.....	18
4.2. Build	18
4.3. Library Use	18
4.4. Macros	18
4.4.1. IOCTL Services.....	19

4.4.2. Registers	19
4.5. Data Types	19
4.6. Functions.....	20
4.6.1. dsi_close()	20
4.6.2. dsi_init()	20
4.6.3. dsi_ioctl().....	21
4.6.4. dsi_open().....	22
4.6.5. dsi_read().....	23
4.7. IOCTL Services	24
4.7.1. DSI_IOCTL_ADC_RATE_OUT	24
4.7.2. DSI_IOCTL_AIN_BUF_CLEAR	25
4.7.3. DSI_IOCTL_AIN_BUF_INPUT	25
4.7.4. DSI_IOCTL_AIN_BUF_OVERFLOW	25
4.7.5. DSI_IOCTL_AIN_BUF_THRESH.....	26
4.7.6. DSI_IOCTL_AIN_BUF_UNDERFLOW	26
4.7.7. DSI_IOCTL_AIN_COUPLING.....	26
4.7.8. DSI_IOCTL_AIN_FILTER	27
4.7.9. DSI_IOCTL_AIN_FILTER_XX_XX.....	27
4.7.10. DSI_IOCTL_AIN_FILTER_SEL	28
4.7.11. DSI_IOCTL_AIN_MODE	28
4.7.12. DSI_IOCTL_AIN_RANGE	28
4.7.13. DSI_IOCTL_AIN_RANGE_XX_XX	29
4.7.14. DSI_IOCTL_AIN_RANGE_SEL	29
4.7.15. DSI_IOCTL_AUTOCAL	30
4.7.16. DSI_IOCTL_CH_GRP_X_SRC.....	30
4.7.17. DSI_IOCTL_CHANNEL_ORDER	31
4.7.18. DSI_IOCTL_CHANNELS_READY	31
4.7.19. DSI_IOCTL_COUPLING_MODE_AC.....	31
4.7.20. DSI_IOCTL_COUPLING_MODE_DC.....	31
4.7.21. DSI_IOCTL_DATA_FORMAT	32
4.7.22. DSI_IOCTL_DATA_WIDTH.....	32
4.7.23. DSI_IOCTL_EXT_CLK_SRC.....	32
4.7.24. DSI_IOCTL_EXT_TRIG	33
4.7.25. DSI_IOCTL_GPS_ENABLE	33
4.7.26. DSI_IOCTL_GPS_LOCKED	33
4.7.27. DSI_IOCTL_GPS_POLARITY	34
4.7.28. DSI_IOCTL_GPS_RATE_LOCKED	34
4.7.29. DSI_IOCTL_GPS_TARGET_RATE.....	34
4.7.30. DSI_IOCTL_GPS_TOLERANCE	34
4.7.31. DSI_IOCTL_INIT_MODE	35
4.7.32. DSI_IOCTL_INITIALIZE	35
4.7.33. DSI_IOCTL_IRQ_SEL	35
4.7.34. DSI_IOCTL_PLL_REF_FREQ_HZ	36
4.7.35. DSI_IOCTL_QUERY	36
4.7.36. DSI_IOCTL_RATE_DIV_X_NDIV	38
4.7.37. DSI_IOCTL_RATE_GEN_X_NRATE.....	39
4.7.38. DSI_IOCTL_RATE_GEN_X_NREF.....	39
4.7.39. DSI_IOCTL_RATE_GEN_X_NVCO.....	39
4.7.40. DSI_IOCTL_REG_MOD.....	40
4.7.41. DSI_IOCTL_REG_READ	40
4.7.42. DSI_IOCTL_REG_WRITE	41
4.7.43. DSI_IOCTL_RX_IO_ABORT.....	41
4.7.44. DSI_IOCTL_RX_IO_MODE.....	41
4.7.45. DSI_IOCTL_RX_IO_OVERFLOW	42

4.7.46. DSI_IOCTL_RX_IO_TIMEOUT	42
4.7.47. DSI_IOCTL_RX_IO_UNDERFLOW	42
4.7.48. DSI_IOCTL_SW_SYNC	43
4.7.49. DSI_IOCTL_SW_SYNC_MODE.....	43
4.7.50. DSI_IOCTL_TA_INV_EXT_TRIGGER.....	43
4.7.51. DSI_IOCTL_THRES_FLAG_CBL	44
4.7.52. DSI_IOCTL_WAIT_CANCEL.....	44
4.7.53. DSI_IOCTL_WAIT_EVENT.....	45
4.7.54. DSI_IOCTL_WAIT_STATUS	47
4.7.55. DSI_IOCTL_XCVR_TYPE.....	48
5. The Driver.....	49
5.1. Files.....	49
5.2. Build	49
5.3. Startup.....	49
5.3.1. Manual Driver Startup Procedures	49
5.3.2. Automatic Driver Startup Procedures.....	50
5.4. Verification	51
5.5. Version.....	52
5.6. Shutdown	52
6. Document Source Code Examples.....	53
6.1. Files.....	53
6.2. Build	53
6.3. Library Use	53
7. Utilities Source Code.....	54
7.1. Files.....	54
7.2. Build	54
7.3. Library Use	54
8. Operating Information	55
8.1. Debugging Aids	55
8.1.1. Device Identification	55
8.1.2. Detailed Register Dump	55
8.2. Analog Input Configuration	55
8.3. Data Transfer Modes.....	56
8.3.1. PIO - Programmed I/O	56
8.3.2. BMDMA - Block Mode DMA	56
8.3.3. DMDMA - Demand Mode DMA	56
8.4. Multi-Board Synchronization	56
8.4.1. Star Configuration	56
8.4.2. Daisy Chain Configuration.....	57
8.5. Clearing the Input Buffer	57
8.5.1. Clear Immediately	58
8.5.2. Clear At a Scan Boundary	58

9. Sample Applications	59
9.1. billion - Billion Byte Read - .../billion/	59
9.2. fref - FREF measurement tool - .../fref/	59
9.3. fsamp - Sample Rate - .../fsamp/	59
9.4. id - Identify Board - .../id/	59
9.5. regs - Register Access - .../regs/	59
9.6. rxrate - Receive Rate - .../rxrate/	59
9.7. savedata - Save Acquired Data - .../savedata/	59
9.8. sbtest - Single Board Test - .../sbtest/	59
9.9. signals - Digital Signals - .../signals/	59
9.10. stream - Stream Rx Data to Disk - .../stream/	60
9.11. sw_sync - Software Sync - .../sw_sync/	60
9.12. mbsync – Multi-Board Sync - .../mbsync/	60
Document History	61

Table of Figures

Figure 1 Basic architectural representation.....	9
Figure 2 The <i>star</i> configuration with three or more boards requires a Clock Driver board.....	57
Figure 3 The <i>star</i> configuration with only two boards does not require a Clock Driver board.	57
Figure 4 In this configuration the clock and sync signals are daisy chained from one board to the next.	57

1. Introduction

1.1. Purpose

The purpose of this document is to describe the interface to the 24DSI 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 24DSI 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
CPCI	Compact PCI
DMA	Direct Memory Access
DMDMA	Demand Mode DMA
GSC	General Standards Corporation
PC104P	This refers to the PC/104+ form factor.
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 24DSI installation directory or any of its subdirectories.
24DSI	This is used as a general reference to any device supported by this driver, which includes the 24DSI32, the 24DSI12, the 24DSI8 and the 24DSI6.
API Library	This is a library that provides application-level access to 24DSI hardware.
Application	This is a user mode process, which runs in user space with user mode privileges.
Driver	This is the 24DSI 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

NOTE: The driver is not able to distinguish a 24DSI8 from a 24DSI12-8. Therefore, the 24DSI8R is reported as a 24DSI12-8, which is an eight-channel version of the 24DSI12.

1.4.1. Basic Software Architecture

This section describes the general architecture for the basic components that comprise 24DSI applications. The overall architecture is illustrated in Figure 1 below.

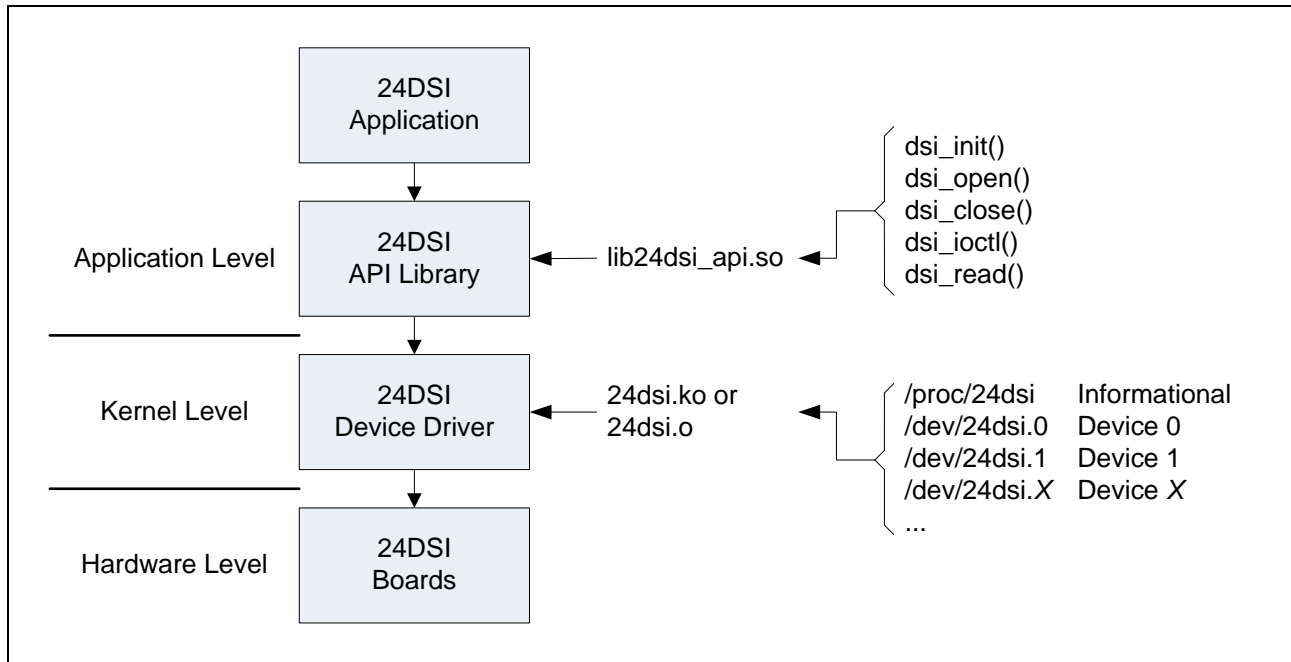


Figure 1 Basic architectural representation.

1.4.2. API Library

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

1.4.3. Device Driver

The device driver is the host software that provides a means of communicating directly with 24DSI 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 24DSI is a high-performance, 24-bit analog input board that incorporates from four to 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 200K samples per second over each channel. Internal clocking permits sampling rates from 200K samples per second down to 2.0K samples per second. For PLL boards the rates programmed are maintained via on-board Phase Locked Loop circuitry. 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 24DSI to sustain continuous throughput from the cable interface independent of the PCI bus interface. The 24DSI also permits multiple boards to be synchronized so that all boards sample data in unison. The board also includes logic to synchronize sampling to a GPS 1PPS input signal. In addition, the board includes autocalibration capability.

1.6. Reference Material

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

- The applicable *24DSI User Manual* from General Standards Corporation.
- The *PCI9056 PCI Bus Master Interface Chip* data handbook from PLX Technology, Inc. †
- The *PCI9080 PCI Bus Master Interface Chip* data handbook from PLX Technology, Inc. †

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

1.7. Licensing

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

2. Installation

2.1. CPU and Kernel Support

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

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

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

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

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

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

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

2.1.1. 32-bit Support Under 64-bit Environments

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

2.2. The `/proc/` File System

While the driver is running, the text file `/proc/24dsi` 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: 4.17.107.50
32-bit support: yes
boards: 1
models: 24DSI12
```

Entry	Description
version	This gives the driver version number in the form <code>x.x.x.x</code> .
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
<code>24dsi.linux.tar.gz</code>	This archive contains the driver, the API Library and all related files.
<code>24dsi_linux_um.pdf</code>	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
<code>24dsi/</code>	This is the driver root directory. It contains the documentation, the Overall Make Script (section 2.7, page 13) and the below listed subdirectories.
<code>.../api/</code>	This directory contains the API Library source files (section 4, page 18).
<code>.../docsrc/</code>	This directory contains the source files for the code samples given in this document (section 6, page 53).
<code>.../driver/</code>	This directory contains the device driver source files (section 5, page 49).
<code>.../include/</code>	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 59).
.../utils/	This directory contains the source files for the utility libraries used by the sample applications (section 7, page 54).

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

```
tar -xzf 24dsi.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 52).
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 24dsi.linux.tar.gz 24dsi
```

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

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

5. If the automatic startup procedure was adopted (section 5.3.2, page 50), then edit the system startup script `rc.local` and remove the line that invokes the 24DSI'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 (`.../24dsi/`).

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/lib24dsi_api.so
Defined and Not Empty	==== Linking: ../lib/lib24dsi_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/24dsi_utils.a
Defined and Not Empty	==== Linking: ../lib/24dsi_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 24DSI 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 24DSI 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 24DSI specific header files. Therefore, sources may include only this one 24DSI header and make files may reference only this one 24DSI include directory.

Description	File	Location
Header File	24dsi_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 24DSI 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 24DSI static library and only this one 24DSI library directory.

Description	File	Location
Static Library	24dsi_main.a	.../lib/
	24dsi_multi.a	

NOTE: For applications using the 24DSI and no other GSC devices, link the 24dsi_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 24DSI API Library is implemented as a shared library and is thus not linked with the 24DSI Main Library. The API Library must be linked with applications by adding the argument -l24dsi_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 24DSI API Library is the software interface between user applications and the 24DSI device driver. The interface is accessed by including the header file `24dsi_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	<code>24dsi_api.h</code>	.../include/
Library File	<code>lib24dsi_api.so</code>	.../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	<code>24dsi_api.h</code>	.../include/	
Shared Object Library	<code>lib24dsi_api.so</code>	.../lib/	
		/usr/lib/	<code>-l24dsi_api</code>

4.4. Macros

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

4.4.2.1. GSC Registers

The following table gives the complete set of GSC specific 24DSI 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 24DSI 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	Information
DSI_GSC_AVR	Autocal Values Register (AVR)	
DSI_GSC_BCFGR	Board Configuration Register (BCFGR)	
DSI_GSC_BCTLR	Board Control Register (BCTLR)	
DSI_GSC_BSR	Buffer Size Register (BSR)	
DSI_GSC_IBCR	Input Buffer Control Register (IBCR)	
DSI_GSC_IDBR	Input Data Buffer Register (IDBR)	
DSI_GSC_GSR	GPS Synchronization Register (GSR)	24DSI12 specific
DSI_GSC_NREFCR	NREF Control Register (NREFCR)	24DSI12 and PLL specific
DSI_GSC_NVCOCR	NVCO Control Register (NVCOCR)	24DSI32 and PLL specific
DSI_GSC_PRFR	PLL Reference Frequency Register (PRFR)	
DSI_GSC_RAR	Rate Assignments Register (RAR)	
DSI_GSC_RCAR	Rate Control A Register (RCAR)	24DSI12 specific
DSI_GSC_RCBR	Rate Control B Register (RCBR)	24DSI12 specific
DSI_GSC_RCR	Rate Control Register (RCR)	24DSI32 and Legacy specific
DSI_GSC_RDR	Rate Divisors Register (RDR)	
DSI_GSC_RFCR	Range and Filter Control Register (RFCR)	24DSI32 specific

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 files `gsc_pci9056.h` and `gsc_pci9080.h`, which are automatically included via `24dsi.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 files `gsc_pci9056.h` and `gsc_pci9080.h`, which are automatically included via `24dsi.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. dsi_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 dsi_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 "24dsi_dsl.h"

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

    ret = dsi_close(fd);

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

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

4.6.2. dsi_init()

This function is the entry point to initializing the 24DSI 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 dsi_init(void);
```

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

Example

```
#include <stdio.h>

#include "24dsi_dsl.h"

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

    ret = dsi_init();

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

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

4.6.3. dsi_ioctl()

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

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

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

Prototype

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

Argument	Description
<code>fd</code>	This is the file descriptor obtained from the open service (section 4.6.4, page 22).
<code>request</code>	This specifies the desired operation to be performed (section 4.7, page 24).
<code>arg</code>	This is specific to the IOCTL operation specified by the <code>request</code> argument.

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

Example

```
#include <stdio.h>

#include "24dsi_dsl.h"

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

    ret = dsi_ioctl(fd, request, arg);

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

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

4.6.4. dsi_open()

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

Prototype

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

Argument	Description						
device	This is the zero-based index of the 24DSI to access. †						
share	Open the device in Shared Access Mode? If non-zero the device is opened in Shared Access Mode (see below). If zero the device is opened in Exclusive Access Mode (see below).						
fd	The device handle is returned here. The pointer cannot be NULL. Values returned are as follows. <table border="1" data-bbox="451 1430 1266 1528"> <tr> <th>Value</th><th>Description</th></tr> <tr> <td>>= 0</td><td>This is the handle to use to access the device in subsequent calls.</td></tr> <tr> <td>-1</td><td>There was an error. The device is not accessible.</td></tr> </table>	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.
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 "24dsi_dsl.h"
```

```

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

    ret = dsi_open(device, share, fd);

    if (ret)
        printf("ERROR: dsi_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. `dsi_read()`

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

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

NOTE: For additional information refer to the Data Transfer Modes section (section 8.3, page 56).

NOTE: Applications may experience improved responsiveness with read requests by coordinating the Buffer Threshold value with the number of samples requested. Refer to the `DSI_IOCTL_AIN_BUF_THRESH` service (section 4.7.5, page 26).

Prototype

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

Argument	Description
<code>fd</code>	This is the file descriptor obtained from the open service (section 4.6.4, page 22).
<code>dst</code>	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 4.7.46, page 42) before the entire request could be satisfied.
< 0	An error occurred. See error value description above.

Example

```
#include <stdio.h>

#include "24dsi_dsl.h"

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

    ret = dsi_read(fd, dst, bytes);

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

4.7.1. DSI_IOCTL_ADC_RATE_OUT

This service permits the ADC sample rate clock to appear at the cable's external clock output signal.

Usage

Argument	Description
request	DSI_IOCTL_ADC_RATE_OUT
arg	s32*

Valid argument values are as follows.

Value	Description
-1	Retrieve the current setting.
DSI_ADC_RATE_OUT_NO	Output the clock designated by the DSI_IOCTL_EXT_CLK_SRC IOCTL service.
DSI_ADC_RATE_OUT_YES	Output the ADC sample rate clock on the external clock output signal.

4.7.2. DSI_IOCTL_AIN_BUF_CLEAR

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

Usage

Argument	Description
request	DSI_IOCTL_AIN_BUF_CLEAR
arg	Not used.

NOTE: With this service the buffer is cleared immediately. This is not timed to occur at a scan boundary and may result in a partial scan being cleared from or entering the buffer. For additional information on clearing the input buffer refer to section 8.5 on page 57.

4.7.3. DSI_IOCTL_AIN_BUF_INPUT

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

Usage

Argument	Description
request	DSI_IOCTL_AIN_BUF_INPUT
arg	s32*

Valid argument values are as follows.

Value	Description
-1	Retrieve the current setting.
DSI_AIN_BUF_INPUT_DISABLE	Disable input to the input buffer.
DSI_AIN_BUF_INPUT_ENABLE	Enable sample data to enter the input buffer.

NOTE: With this service the buffer input stream is affected immediately. This is not timed to occur at a scan boundary and may result in a partial scan entering the buffer. For additional information on clearing the input buffer refer to section 8.5 on page 57.

4.7.4. DSI_IOCTL_AIN_BUF_OVERFLOW

This service operates on the Analog Input Overflow status.

Usage

Argument	Description
request	DSI_IOCTL_AIN_BUF_OVERFLOW
arg	s32*

Valid argument values are as follows.

Value	Description
DSI_AIN_BUF_OVERFLOW_CLEAR	Clear the overflow status.
DSI_AIN_BUF_OVERFLOW_TEST	Report if an overflow has occurred.

Valid returned values are as follows.

Value	Description
DSI_AIN_BUF_OVERFLOW_NO	An overflow did not occur.
DSI_AIN_BUF_OVERFLOW_YES	An overflow did occur.

4.7.5. DSI_IOCTL_AIN_BUF_THRESH

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

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

Usage

Argument	Description
request	DSI_IOCTL_AIN_BUF_THRESH
arg	s32*

Valid argument values are from zero to 0x3FFFFF, and -1. A value of -1 will return the current threshold level setting.

4.7.6. DSI_IOCTL_AIN_BUF_UNDERFLOW

This service operates on the Analog Input Underflow status.

Usage

Argument	Description
request	DSI_IOCTL_AIN_BUF_UNDERFLOW
arg	s32*

Valid argument values are as follows.

Value	Description
DSI_AIN_BUF_UNDERFLOW_CLEAR	Clear the underflow status.
DSI_AIN_BUF_UNDERFLOW_TEST	Report if an underflow has occurred.

Valid returned values are as follows.

Value	Description
DSI_AIN_BUF_UNDERFLOW_NO	An underflow did not occur.
DSI_AIN_BUF_UNDERFLOW_YES	An underflow did occur.

4.7.7. DSI_IOCTL_AIN_COUPLING

This service selects AC or DC coupling on the analog input signals.

Usage

Argument	Description
request	DSI_IOCTL_AIN_COUPLING
arg	s32*

Valid argument values are as follows.

Value	Description
-1	Retrieve the current setting.
DSI_AIN_COUPLING_AC	Select AC coupling.
DSI_AIN_COUPLING_DC	Select DC coupling.

4.7.8. DSI_IOCTL_AIN_FILTER

This service selects low or high frequency image input filtering.

NOTE: For those boards which support filter assignments to different channel ranges, this global filter assigning is ignored when the selective assignment option is enabled. Refer to the DSI_IOCTL_AIN_FILTER_SEL service of section 4.7.10 on page 28.

Usage

Argument	Description
request	DSI_IOCTL_AIN_FILTER
arg	s32*

Valid argument values are as follows.

Value	Description
-1	Retrieve the current setting.
DSI_AIN_FILTER_HI	Select high frequency filtering.
DSI_AIN_FILTER_LOW	Select low frequency filtering.

4.7.9. DSI_IOCTL_AIN_FILTER_XX_XX

This service selects low or high frequency image filtering for the respective group of channels. The specified option takes affect only when selective assignments is enabled, and then, only when the board supports the selective assignment feature.

Value	Description
DSI_IOCTL_AIN_FILTER_00_03	This refers to channels 0 through 3.
DSI_IOCTL_AIN_FILTER_04_07	This refers to channels 4 through 7.
DSI_IOCTL_AIN_FILTER_08_11	This refers to channels 8 through 11.
DSI_IOCTL_AIN_FILTER_12_15	This refers to channels 12 through 15.
DSI_IOCTL_AIN_FILTER_16_19	This refers to channels 16 through 19.
DSI_IOCTL_AIN_FILTER_20_23	This refers to channels 20 through 23.
DSI_IOCTL_AIN_FILTER_24_27	This refers to channels 24 through 27.
DSI_IOCTL_AIN_FILTER_28_31	This refers to channels 28 through 31.

NOTE: Selective filter assignments supersede the global assignment when the selective assignment option is enabled. Refer to the DSI_IOCTL_AIN_FILTER_SEL service of section 4.7.10 on page 28.

Usage

Argument	Description
request	DSI_IOCTL_AIN_FILTER_XX_XX
arg	s32*

Valid argument values are as follows.

Value	Description
-1	Retrieve the current setting.
IMAGE_FILTER_HI_FREQ	Select high frequency filtering.
IMAGE_FILTER_LO_FREQ	Select low frequency filtering.

4.7.10. DSI_IOCTL_AIN_FILTER_SEL

This service enables or disables selective filter assignments, for those boards which support selective assignments. When enabled, this feature supersedes the global setting of the DSI_IOCTL_AIN_FILTER service of section 4.7.8 on page 27.

Usage

Argument	Description
request	DSI_IOCTL_AIN_FILTER_SEL
arg	s32*

Valid argument values are as follows.

Value	Description
-1	Retrieve the current setting.
DSI_AIN_FILTER_SEL_DISABLE	This disables selective filter assignments.
DSI_AIN_FILTER_SEL_ENABLE	This enables selective filter assignments.

4.7.11. DSI_IOCTL_AIN_MODE

This service selects the analog input signal configuration mode.

Usage

Argument	Description
request	DSI_IOCTL_AIN_MODE
arg	s32*

Valid argument values are as follows.

Value	Description
-1	Retrieve the current setting.
DSI_AIN_MODE_DIFF	Configure the input channels for differential operation.
DSI_AIN_MODE_VREF	Connect the input channels to the onboard VREF signal.
DSI_AIN_MODE_ZERO	Connect the input channels to the onboard zero voltage signal.

4.7.12. DSI_IOCTL_AIN_RANGE

This service sets the analog input voltage range.

Usage

Argument	Description
request	DSI_IOCTL_AIN_RANGE
arg	s32*

Valid argument values are as follows.

Value	Description
-1	Retrieve the current setting.
DSI_AIN_RANGE_2_5V	Set the input voltage range to ± 2.5 volts.
DSI_AIN_RANGE_5V	Set the input voltage range to ± 5 volts.
DSI_AIN_RANGE_10V	Set the input voltage range to ± 10 volts.

NOTE: The ± 10 volts option is not available for low power boards.

4.7.13. DSI_IOCTL_AIN_RANGE_XX_XX

This service selects the voltage range for the respective group of channels. The specified option takes affect only when selective assignments is enabled, and then, only when the board supports the selective assignment feature.

Value	Description
DSI_IOCTL_AIN_RANGE_00_03	This refers to channels 0 through 3.
DSI_IOCTL_AIN_RANGE_04_07	This refers to channels 4 through 7.
DSI_IOCTL_AIN_RANGE_08_11	This refers to channels 8 through 11.
DSI_IOCTL_AIN_RANGE_12_15	This refers to channels 12 through 15.
DSI_IOCTL_AIN_RANGE_16_19	This refers to channels 16 through 19.
DSI_IOCTL_AIN_RANGE_20_23	This refers to channels 20 through 23.
DSI_IOCTL_AIN_RANGE_24_27	This refers to channels 24 through 27.
DSI_IOCTL_AIN_RANGE_28_31	This refers to channels 28 through 31.

NOTE: Selective voltage range assignments supersede the global assignment when the selective assignment option is enabled. Refer to the DSI_IOCTL_AIN_RANGE_SEL service of section 4.7.14 on page 29.

Usage

Argument	Description
request	DSI_IOCTL_AIN_RANGE_XX_XX
arg	s32*

Valid argument values are as follows.

Value	Description
-1	Retrieve the current setting.
DSI_AIN_RANGE_2_5V	Set the input voltage range to ± 2.5 volts.
DSI_AIN_RANGE_5V	Set the input voltage range to ± 5 volts.
DSI_AIN_RANGE_10V	Set the input voltage range to ± 10 volts.

NOTE: The ± 10 volts option is not available for low power boards.

4.7.14. DSI_IOCTL_AIN_RANGE_SEL

This service enables or disables selective voltage range assignments, for those boards which support selective assignments. When enabled, this feature supersedes the global setting of the DSI_IOCTL_AIN_RANGE service of section 4.7.12 on page 28.

Usage

Argument	Description
request	DSI_IOCTL_AIN_RANGE_SEL
arg	s32*

Valid argument values are as follows.

Value	Description
-1	Retrieve the current setting.
DSI_AIN_RANGE_SEL_DISABLE	This disables selective assignments.
DSI_AIN_RANGE_SEL_ENABLE	This enables selective assignments.

4.7.15. DSI_IOCTL_AUTOCAL

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

WARNING: Do not access the board while an autocalibration cycle is in progress. Doing so may produce indeterminate results, and may lockup the board.

NOTE: This service overwrites the current interrupt selection in order to detect the Autocalibration Done interrupt.

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

Usage

Argument	Description
request	DSI_IOCTL_AUTOCAL
arg	Not used.

4.7.16. DSI_IOCTL_CH_GRP_X_SRC

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

Value	Description
DSI_IOCTL_CH_GRP_0_SRC	This refers to Channel Group 0.
DSI_IOCTL_CH_GRP_1_SRC	This refers to Channel Group 1.
DSI_IOCTL_CH_GRP_2_SRC	This refers to Channel Group 2.
DSI_IOCTL_CH_GRP_3_SRC	This refers to Channel Group 3.

Usage

Argument	Description
request	DSI_IOCTL_CH_GRP_X_SRC
arg	s32*

Valid argument values are as follows.

Value	Description
-1	Retrieve the current setting.
DSI_CH_GRP_SRC_GEN_A	This sets the source to Rate Generator A.
DSI_CH_GRP_SRC_GEN_B	This sets the source to Rate Generator B.
DSI_CH_GRP_SRC_EXTERN	This sets the source to External option.
DSI_CH_GRP_SRC_DIR_EXTERN	This sets the source to Direct External option.
DSI_CH_GRP_SRC_DISABLE	This disables the respective channel group.
DSI_CH_GRP_SRC_ENABLE	This enables the respective channel group.

Some options are specific to particular board versions. Refer to your hardware user manual for details.

4.7.17. DSI_IOCTL_CHANNEL_ORDER

This service selects between synchronous and asynchronous channel scanning, which is the order the channel data enters the analog input buffer.

Usage

Argument	Description
request	DSI_IOCTL_CHANNEL_ORDER
arg	s32*

Valid argument values are as follows.

Value	Description
-1	Retrieve the current setting.
DSI_CHANNEL_ORDER_ASYNC	Select asynchronous scanning.
DSI_CHANNEL_ORDER_SYNC	Select synchronous scanning.

4.7.18. DSI_IOCTL_CHANNELS_READY

This service waits for the Channels Ready status to be asserted. The service waits no longer than the read I/O timeout period.

Usage

Argument	Description
request	DSI_IOCTL_CHANNELS_READY
arg	Not used.

4.7.19. DSI_IOCTL_COUPLING_MODE_AC

This service configures the input signals for AC coupling.

Usage

Argument	Description
request	DSI_IOCTL_COUPLING_MODE_AC
arg	s32*

Valid argument values consist of any bitmap value referencing existing board channels, or -1 to retrieve the current setting. The lowest order bit refers to channel 0. If a bit is set, then the channel is configured for AC coupling.

4.7.20. DSI_IOCTL_COUPLING_MODE_DC

This service configures the input signals for DC coupling.

Usage

Argument	Description
request	DSI_IOCTL_COUPLING_MODE_DC
arg	s32*

Valid argument values consist of any bitmap value referencing existing board channels, or -1 to retrieve the current setting. The lowest order bit refers to channel 0. If a bit is set, then the channel is configured for DC coupling.

4.7.21. DSI_IOCTL_DATA_FORMAT

This service sets the data encoding format.

Usage

Argument	Description
request	DSI_IOCTL_DATA_FORMAT
arg	s32*

Valid argument values are as follows.

Value	Description
-1	Retrieve the current setting.
DSI_DATA_FORMAT_OFF_BIN	Select the Offset Binary encoding format.
DSI_DATA_FORMAT_2S_COMP	Select the Twos Complement data format.

4.7.22. DSI_IOCTL_DATA_WIDTH

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

Usage

Argument	Description
request	DSI_IOCTL_DATA_WIDTH
arg	s32*

Valid argument values are as follows.

Value	Description
-1	Retrieve the current setting.
DSI_DATA_WIDTH_16	Convert sampled data with 16-bits of resolution.
DSI_DATA_WIDTH_18	Convert sampled data with 18-bits of resolution.
DSI_DATA_WIDTH_20	Convert sampled data with 20-bits of resolution.
DSI_DATA_WIDTH_24	Convert sampled data with 24-bits of resolution.

4.7.23. DSI_IOCTL_EXT_CLK_SRC

This service configures the source for the external clock output.

Usage

Argument	Description
request	DSI_IOCTL_EXT_CLK_SRC
arg	s32*

Valid argument values are as follows.

Value	Description
-1	Retrieve the current setting.
DSI_EXT_CLK_SRC_GEN_A	This selects Rate Generator A.
DSI_EXT_CLK_SRC_GRP_0	This selects the Channel Group 0 sample clock.

4.7.24. DSI_IOCTL_EXT_TRIG

This service configures the board's external burst trigger input feature, when supported.

Usage

Argument	Description
request	DSI_IOCTL_EXT_TRIG
arg	s32*

Valid argument values are as follows.

Value	Description
-1	Retrieve the current setting.
DSI_EXT_TRIG_ARM	This arms the trigger.
DSI_EXT_TRIG_DISARM	This disarms the trigger.

4.7.25. DSI_IOCTL_GPS_ENABLE

This service enables or disables the GPS Synchronization feature, when supported by the board.

Usage

Argument	Description
request	DSI_IOCTL_GPS_ENABLE
arg	s32*

Valid argument values are as follows.

Value	Description
-1	Retrieve the current setting.
DSI_GPS_ENABLE_NO	Disable GPS Synchronization.
DSI_GPS_ENABLE_YES	Enable GPS Synchronization.

4.7.26. DSI_IOCTL_GPS_LOCKED

This service query's the board for GPS Locked status, on those boards which support GPS Synchronization.

Usage

Argument	Description
request	DSI_IOCTL_GPS_LOCKED
arg	s32*

Valid argument values are as follows.

Value	Description
-1	Retrieve the current setting.
DSI_GPS_LOCKED_NO	The board is not synchronized to the GPS input signal.
DSI_GPS_LOCKED_YES	The board is synchronized to the GPS input signal.

4.7.27. DSI_IOCTL_GPS_POLARITY

This service configures the board for the polarity of the GPS input signal, for those boards which support GPS Synchronization.

Usage

Argument	Description
request	DSI_IOCTL_GPS_POLARITY
arg	s32*

Valid argument values are as follows.

Value	Description
-1	Retrieve the current setting.
DSI_GPS_POLARITY_NEG	Respond to a high-to-low going signal edge.
DSI_GPS_POLARITY_POS	Respond to a low-to-high going signal edge.

4.7.28. DSI_IOCTL_GPS_RATE_LOCKED

This service query's the board for the GPS Sampling Rate Locked status, on those boards which support GPS Synchronization.

Usage

Argument	Description
request	DSI_IOCTL_GPS_RATE_LOCKED
arg	s32*

Valid argument values are as follows.

Value	Description
-1	Retrieve the current setting.
DSI_GPS_RATE_LOCKED_NO	The GPS feature has not locked to data sampling.
DSI_GPS_RATE_LOCKED_YES	The GPS feature has locked to data sampling.

4.7.29. DSI_IOCTL_GPS_TARGET_RATE

This service sets the data sampling rate that the GPS feature monitors, for those boards which support GPS Synchronization.

Usage

Argument	Description
request	DSI_IOCTL_GPS_TARGET_RATE
arg	s32*

Valid argument values are from zero to 0x3FFFF, and -1. The value -1 is used to retrieve the current setting.

4.7.30. DSI_IOCTL_GPS_TOLERANCE

This service sets the GPS feature's sample rate deviation tolerance, for those boards which support GPS Synchronization.

Usage

Argument	Description
request	DSI_IOCTL_GPS_TOLERANCE
arg	s32*

Valid argument values are as follows.

Value	Description
-1	Retrieve the current setting.
DSI_GPS_TOLERANCE_NARROW	This selects narrow tolerance. †
DSI_GPS_TOLERANCE_WIDE	This selects wide tolerance. †

† Refer to the hardware user manual for the specific definition of narrow and wide.

4.7.31. DSI_IOCTL_INIT_MODE

This service sets the initiator mode for synchronized, multi-board data acquisition.

Usage

Argument	Description
request	DSI_IOCTL_INIT_MODE
arg	s32*

Valid argument values are as follows.

Value	Description
-1	Retrieve the current setting.
DSI_INIT_MODE_INITIATOR	Operate the board in Initiator Mode.
DSI_INIT_MODE_TARGET	Operate the board in Target Mode.

4.7.32. DSI_IOCTL_INITIALIZE

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

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

Usage

Argument	Description
request	DSI_IOCTL_INITIALIZE
arg	Not used.

4.7.33. DSI_IOCTL_IRQ_SEL

This service selects which interrupt option is active for the firmware interrupt.

Usage

Argument	Description
request	DSI_IOCTL_IRQ_SEL
arg	s32*

Valid argument values are as follows.

Value	Description
-1	Retrieve the current setting.
DSI_IRQ_AIN_BUF_THRESH_H2L	This refers to a high-to-low transition of the input buffer threshold flag.
DSI_IRQ_AIN_BUF_THRESH_L2H	This refers to a low-to-high transition of the input buffer threshold flag.
DSI_IRQ_AUTOCAL_DONE	This refers to Autocalibration completion.
DSI_IRQ_CHAN_READY	This refers to assertion of the Channel Ready status.
DSI_IRQ_INIT_DONE	This refers to initialization completion.

4.7.34. DSI_IOCTL_PLL_REF_FREQ_HZ

This service retrieves the value of the PLL Frequency Reference Register. Refer to the hardware user manual for the usage of this register. This service is applicable to PLL version board only.

Usage

Argument	Description
request	DSI_IOCTL_PLL_REF_FREQ_HZ
arg	s32*

4.7.35. DSI_IOCTL_QUERY

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

Usage

Argument	Description
request	DSI_IOCTL_QUERY
arg	s32*

Valid argument values are as follows.

Value	Description
DSI_QUERY_AUTOCAL_MS	This returns the maximum duration of the Autocalibration cycle in milliseconds. If this is zero, then the board doesn't support autocalibration.
DSI_QUERY_CH_GRP_0_RAR	Does the Rate Assignment Register use the Channel Group 0 assignment as a master setting? (0 = no, 1 = yes)
DSI_QUERY_CHANNEL_GPS	This returns the number of input channel groups.
DSI_QUERY_CHANNEL_MAX	This returns the maximum number of input channels supported by all boards of the same model as the board accessed.
DSI_QUERY_CHANNEL_QTY	This returns the actual number of input channels on the current board.
DSI_QUERY_COUNT	This returns the number of query options supported by the IOCTL service.
DSI_QUERY_D0_1_IN_MODE	Do Board Control Register bits D0 and D1 control the Analog Input Mode? (0 = no, 1 = yes)
DSI_QUERY_D2_3_RANGE	Do Board Control Register bits D2 and D3 control the Voltage Range? (0 = no, 1 = yes)
DSI_QUERY_D15_PLL	Does Board Configuration Register bit D15 pertain to PLL support? (0 = no, 1 = yes)

DSI_QUERY_D16_CHANNELS	This returns the number of input channels that Board Configuration Register bit D16 pertain to when the bit is set. A value of zero indicates that this bit does not have this meaning.
DSI_QUERY_D16_CHANNELS_0	This returns the number of input channels that Board Configuration Register bit D16 pertain to when the bit is clear. If zero, then no specific channel count is specified.
DSI_QUERY_D16_SYNC_1	Does the value “1” in Board Control Register bit D16 pertain to Synchronous? (1 = yes, 0 = no) If yes, then “0” means Asynchronous. If no, then the value meanings are reversed.
DSI_QUERY_D17_CHANNELS	This returns the number of input channels that Board Configuration Register bit D17 pertain to when the bit is set. A value of zero indicates that this bit does not have this meaning.
DSI_QUERY_D17_CHANNELS_0	This returns the number of input channels that Board Configuration Register bit D17 pertain to when the bit is clear. If zero, then no specific channel count is specified.
DSI_QUERY_D17_18_RANGE	Do Board Configuration Register bits D18 and D18 report the board’s hard coded voltage range? (0 = no, 1 = yes)
DSI_QUERY_D18_CF_FILT	Does Board Configuration Register bit D18 pertain to Custom Frequency Filters? (0 = no, 1 = yes)
DSI_QUERY_D18_RIO	Does Board Configuration Register bit D18 pertain to Rear-Panel I/O? (0 = no, 1 = yes)
DSI_QUERY_D19_EXT_TEMP	Does Board Configuration Register bit D19 pertain to Extended Temperature operation? (0 = no, 1 = yes)
DSI_QUERY_D19_FREQ_FILT	Does Board Control Register bit D19 control the filter frequency? (0 = no, 1 = yes)
DSI_QUERY_D19_20_FILT	Do Board Configuration Register bits D19 and D20 report the board’s hard coded filter frequency? (0 = no, 1 = yes)
DSI_QUERY_D20_LOW_POWER	Does Board Configuration Register bit D20 pertain to Low Power operation? (0 = no, 1 = yes) The configurable ± 10 -volt range is not available with low power boards.
DSI_QUERY_D20_XCVR	Does Board Control Register bit D20 pertain to TTL/LVDS Transceiver selection? (0 = no, 1 = yes)
DSI_QUERY_D21_EXT_TEMP	Does Board Configuration Register bit D21 report the board’s temperature range configuration? (0 = no, 1 = yes)
DSI_QUERY_D21_EXT_TRIG	Does Board Control Register bit D21 pertain to External Burst Trigger Arming? (0 = no, 1 = yes)
DSI_QUERY_D22_COUPLING	Does Board Control Register bit D22 control AC/DC coupling? (0 = no, 1 = yes)
DSI_QUERY_D22_THR_FLAG	Does Board Control Register bit D21 pertain to Threshold Flag Routing? (0 = no, 1 = yes)
DSI_QUERY_D22_24DSI6LN	Does Board Configuration Register bit D22 report indicate if the board is a24DSI6LN? (0 = no, 1 = yes)
DSI_QUERY_D23_INV_EXT_TRIG	Does Board Control Register bit D23 function as the Invert External Trigger for Triggered Acquisition? (0 = no, 1 = yes)
DSI_QUERY_D23_RATE_OUT	Does Board Control Register bit D23 control the appearance of the ADC clock at cable’s output clock signal? (0 = no, 1 = yes)
DSI_QUERY_DEVICE_TYPE	This returns the identifier value for the board’s type. The value is a member of the <code>gsc_dev_type_t</code> enumeration, which is defined in <code>gsc_common.h</code> .
DSI_QUERY_FGEN_MAX	This returns the maximum supported FGEN value.
DSI_QUERY_FGEN_MIN	This returns the minimum supported FGEN value.
DSI_QUERY_FIFO_SIZE	This returns the size of the input buffer in samples.
DSI_QUERY_FREF_DEFAULT	This gives the default FREF value in Hz.
DSI_QUERY_FSAMP_DEFAULT	This gives the default FSAMP value in S/S.

DSI_QUERY_FSAMP_MAX	This gives the maximum FSAMP value in S/S.
DSI_QUERY_FSAMP_MIN	This gives the minimum FSAMP value in S/S.
DSI_QUERY_GPS_PRESENT	Does the board support the GPS feature? (0 = no, 1 = yes)
DSI_QUERY_INIT_MS	This returns the duration of a board initialization in milliseconds.
DSI_QUERY_LOW_POWER	Is this a Low Power board? (0 = no, 1 = yes)
DSI_QUERY_NDIV_MASK	This returns the mask for the board's NDIV fields.
DSI_QUERY_NDIV_MAX	This returns the maximum supported NDIV value.
DSI_QUERY_NDIV_MIN	This returns the minimum supported NDIV value.
DSI_QUERY_NRATE_MASK	This returns the mask for the board's NRATE fields. This is zero for PLL boards.
DSI_QUERY_NRATE_MAX	This returns the maximum supported NRATE value. This is zero for PLL boards.
DSI_QUERY_NRATE_MIN	This returns the minimum supported NRATE value. This is zero for PLL boards.
DSI_QUERY_NREF_MASK	This returns the mask for the board's NREF fields. This is zero for non-PLL boards.
DSI_QUERY_NREF_MAX	This returns the maximum supported NREF value. This is zero for non-PLL boards.
DSI_QUERY_NREF_MIN	This returns the minimum supported NREF value. This is zero for non-PLL boards.
DSI_QUERY_NVCO_MASK	This returns the mask for the board's NVCO fields. This is zero for non-PLL boards.
DSI_QUERY_NVCO_MAX	This returns the maximum supported NVCO value. This is zero for non-PLL boards.
DSI_QUERY_NVCO_MIN	This returns the minimum supported NVCO value. This is zero for non-PLL boards.
DSI_QUERY_PLL_PRESENT	Is this a PLL board? (0 = no, 1 = yes) If not, then it is a Legacy or Non-PLL board.
DSI_QUERY_RATE_DIV_QTY	This returns the number of Rate Dividers on the board.
DSI_QUERY_RATE_GEN_QTY	This returns the number of Rate Generators on the board.
DSI_QUERY_RCAR_RCBR	Does the board support the Rate Control A Register and the Rate Control B Register? (0 = no, 1 = yes)
DSI_QUERY_REG_ICMR	Does the board support the Input Coupling Mode Register? (0 = no, 1 = yes)
DSI_QUERY_RFCR	Does the board support the Range and Filter Control Register? (0 = no, 1 = yes)

4.7.36. DSI_IOCTL_RATE_DIV_X_NDIV

This service sets the NDIV value for the respective Rate Divider.

Service	Description
DSI_IOCTL_RATE_DIV_0_NDIV	This sets NDIV for Rate Divider 0.
DSI_IOCTL_RATE_DIV_1_NDIV	This sets NDIV for Rate Divider 1.

Usage

Argument	Description
request	DSI_IOCTL_RATE_DIV_X_NDIV
arg	s32*

Valid argument values are those defined in the user manual for NDIV, as well as -1. The documented limits can be obtained at runtime using the DSI_IOCTL_QUERY service with the DSI_QUERY_NDIV_MAX and

DSI_QUERY_NDIV_MIN options. Refer to section 4.7.35 on page 36. Using the value -1 will return the current setting.

4.7.37. DSI_IOCTL_RATE_GEN_X_NRATE

This service sets the NRATE value for the respective Rate Generator for non-PLL version boards.

Service	Description
DSI_IOCTL_RATE_GEN_A_NRATE	This sets NRATE for Rate Generator A.
DSI_IOCTL_RATE_GEN_B_NRATE	This sets NRATE for Rate Generator B.

Usage

Argument	Description
request	DSI_IOCTL_RATE_GEN_X_NRATE
arg	s32*

Valid argument values are those defined in the user manual for NRATE, as well as -1. The documented limits can be obtained at runtime using the DSI_IOCTL_QUERY service with the DSI_QUERY_NRATE_MAX and DSI_QUERY_NRATE_MIN options. Refer to section 4.7.35 on page 36. Using the value -1 will return the current setting.

4.7.38. DSI_IOCTL_RATE_GEN_X_NREF

This service sets the NREF value for the respective Rate Generator for PLL version boards.

Service	Description
DSI_IOCTL_RATE_GEN_A_NREF	This sets NREF for Rate Generator A.
DSI_IOCTL_RATE_GEN_B_NREF	This sets NREF for Rate Generator B.

Usage

Argument	Description
request	DSI_IOCTL_RATE_GEN_X_NREF
arg	s32*

Valid argument values are those defined in the user manual for NREF, as well as -1. The documented limits can be obtained at runtime using the DSI_IOCTL_QUERY service with the DSI_QUERY_NREF_MAX and DSI_QUERY_NREF_MIN options. Refer to section 4.7.35 on page 36. Using the value -1 will return the current setting.

4.7.39. DSI_IOCTL_RATE_GEN_X_NVCO

This service sets the NVCO value for the respective Rate Generator for PLL version boards.

Service	Description
DSI_IOCTL_RATE_GEN_A_NVCO	This sets NVCO for Rate Generator A.
DSI_IOCTL_RATE_GEN_B_NVCO	This sets NVCO for Rate Generator B.

Usage

Argument	Description
request	DSI_IOCTL_RATE_GEN_X_NVCO
arg	s32*

Valid argument values are those defined in the user manual for NVCO, as well as -1. The documented limits can be obtained at runtime using the `DSI_IOCTL_QUERY` service with the `DSI_QUERY_NVCO_MAX` and `DSI_QUERY_NVCO_MIN` options. Refer to section 4.7.35 on page 36. Using the value -1 will return the current setting.

4.7.40. DSI_IOCTL_REG_MOD

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

Usage

Argument	Description
request	DSI_IOCTL_REG_MOD
arg	<code>gsc_reg_t*</code>

Definition

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

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

4.7.41. DSI_IOCTL_REG_READ

This service reads the value of a 24DSI register. This includes the PCI registers, the PLX Feature Set Registers and the GSC firmware registers. Refer to `24dsi.h` and `gsc_pci9080.h` for the complete list of accessible registers.

Usage

Argument	Description
request	DSI_IOCTL_REG_READ
arg	<code>gsc_reg_t*</code>

Definition

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


Fields	Description
reg	This is set to the identifier for the register to access.
value	This is the value read from the specified register.
mask	This is ignored for read request.

4.7.42. DSI_IOCTL_REG_WRITE

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

Usage

Argument	Description
request	DSI_IOCTL_REG_WRITE
arg	<code>gsc_reg_t*</code>

Definition

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

Fields	Description
reg	This is set to the identifier for the register to access.
value	This is the value to write to the specified register.
mask	This is ignored for write request.

4.7.43. DSI_IOCTL_RX_IO_ABORT

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

Usage

Argument	Description
request	DSI_IOCTL_RX_IO_ABORT
arg	<code>s32*</code>

The results are reported as one of the following values.

Value	Description
DSI_IO_ABORT_NO	A <code>read()</code> request was not aborted as none were ongoing.
DSI_IO_ABORT_YES	An ongoing <code>read()</code> request was aborted.

4.7.44. DSI_IOCTL_RX_IO_MODE

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

NOTE: Applications may experience improved responsiveness with read requests by coordinating the Buffer Threshold with the number of samples requested. Refer to the `DSI_IOCTL_AIN_BUF_THRESH` service of section 4.7.5 on page 26.

Usage

Argument	Description
request	DSI_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.

4.7.45. DSI_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	DSI_IOCTL_RX_IO_OVERFLOW
arg	s32*

Valid argument values are as follows.

Value	Description
-1	Retrieve the current setting.
DSI_IO_OVERFLOW_IGNORE	Do not perform the check.
DSI_IO_OVERFLOW_CHECK	Perform the check.

4.7.46. DSI_IOCTL_RX_IO_TIMEOUT

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

Usage

Argument	Description
request	DSI_IOCTL_RX_IO_TIMEOUT
arg	s32*

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

4.7.47. DSI_IOCTL_RX_IO_UNDERFLOW

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

Usage

Argument	Description
request	DSI_IOCTL_RX_IO_UNDERFLOW
arg	s32*

Valid argument values are as follows.

Value	Description
-1	Retrieve the current setting.
DSI_IO_UNDERFLOW_CHECK	Perform the check.
DSI_IO_UNDERFLOW_IGNORE	Do not perform the check.

4.7.48. DSI_IOCTL_SW_SYNC

This service initiates a Software Sync output pulse. The pulse is output by the board only if it is configured as an initiator. The result of issuing a sync pulse is dependent on the `DSI_IOCTL_SW_SYNC_MODE` setting (refer to section 4.7.49 on page 43). When initiating this operation, it is the application's responsibility to wait for the Channel Ready bit to be asserted.

Usage

Argument	Description
request	DSI_IOCTL_SW_SYNC
arg	Not used.

4.7.49. DSI_IOCTL_SW_SYNC_MODE

This service sets the context of the Software Sync control bit.

Usage

Argument	Description
request	DSI_IOCTL_SW_SYNC_MODE
arg	s32*

Valid argument values are as follows.

Value	Description
-1	Retrieve the current setting.
DSI_SW_SYNC_MODE_CLR_BUF	Clear the input buffer when there is a Software Sync request. The clearing of the buffer is timed to occur on a scan boundary. Refer to section 8.5 on page 57 for addition buffer clearing information.
DSI_SW_SYNC_MODE_SW_SYNC	Synchronize input channel scanning when there is a Software Sync request.

4.7.50. DSI_IOCTL_TA_INV_EXT_TRIGGER

This service controls the inversion of the External Trigger signal sensing when operating in Triggered Acquisition mode.

Usage

Argument	Description
request	DSI_IOCTL_TA_INV_EXT_TRIGGER
arg	s32*

Valid argument values are as follows.

Value	Description
-1	Retrieve the current setting.
DSI_TA_INV_EXT_TRIGGER_NO	Do not invert trigger sensing. Trigger on rising edges.
DSI_TA_INV_EXT_TRIGGER_YES	Invert trigger sensing. Trigger on falling edges.

4.7.51. DSI_IOCTL_THRES_FLAG_CBL

This service controls the appearance of the Threshold Flat on the cable interface, for those boards which support the feature.

Usage

Argument	Description
request	DSI_IOCTL_THRES_FLAG_CBL
arg	s32*

Valid argument values are as follows.

Value	Description
-1	Retrieve the current setting.
DSI_THRES_FLAG_CBL_OFF	Route the Auxiliary Input to the Auxiliary Output.
DSI_THRES_FLAG_CBL_OUT	Route the Threshold Flag to the Auxiliary Output.

4.7.52. DSI_IOCTL_WAIT_CANCEL

This service resumes all threads blocked via DSI_IOCTL_WAIT_EVENT IOCTL calls (section 4.7.53, page 45), 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	DSI_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 4.7.53.2 on page 46.
gsc	This specifies the set of DSI_WAIT_GSC_* events whose wait requests are to be cancelled. Refer to section 4.7.53.3 on page 46.
alt	This is unused by the 24DSI driver and should be zero.
io	This specifies the set of DSI_WAIT_IO_* events whose wait requests are to be cancelled. Refer to section 4.7.53.4 on page 47.
timeout_ms	This is unused by wait cancel operations.
count	Upon return this indicates the number of waits that were cancelled.

4.7.53. DSI_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	DSI_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 4.7.53.1 on page 46.
main	This specifies any number of GSC_WAIT_MAIN_* events that the thread is to wait for. Refer to section 4.7.53.2 on page 46.

<code>gsc</code>	This specifies any number of <code>DSI_WAIT_GSC_*</code> events that the thread is to wait for. Refer to section 4.7.53.3 on page 46.
<code>alt</code>	This is unused by the 24DSI driver and must be zero.
<code>io</code>	This specifies any number of <code>DSI_WAIT_IO_*</code> events that the thread is to wait for. Refer to section 4.7.53.4 on page 47.
<code>timeout_ms</code>	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.
<code>count</code>	This is unused by wait event operations and must be zero.

4.7.53.1. `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
<code>GSC_WAIT_FLAG_CANCEL</code>	The wait request was cancelled.
<code>GSC_WAIT_FLAG_DONE</code>	One of the referenced events occurred.
<code>GSC_WAIT_FLAG_TIMEOUT</code>	The timeout period lapsed before a referenced event occurred.

4.7.53.2. `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 24DSI and other General Standards products.

Fields	Description
<code>GSC_WAIT_MAIN_DMA0</code>	This refers to the DMA Done interrupt on DMA engine number zero.
<code>GSC_WAIT_MAIN_DMA1</code>	This refers to the DMA Done interrupt on DMA engine number one.
<code>GSC_WAIT_MAIN_GSC</code>	This refers to any of the Interrupt Control/Status Register interrupts.
<code>GSC_WAIT_MAIN_OTHER</code>	This generally refers to an interrupt generated by another device sharing the same interrupt as the 24DSI.
<code>GSC_WAIT_MAIN_PCI</code>	This refers to any interrupt generated by the 24DSI.
<code>GSC_WAIT_MAIN_SPURIOUS</code>	This refers to board interrupts which should never be generated.
<code>GSC_WAIT_MAIN_UNKNOWN</code>	This refers to board interrupts whose source could not be identified.

4.7.53.3. `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 `DSI_IOCTL_IRQ_SEL` (section 4.7.33, page 35).

Value	Description
<code>DSI_WAIT_GSC_AIN_BUF_THRESH_H2L</code>	This refers to a high-to-low transition of the input buffer threshold flag.
<code>DSI_WAIT_GSC_AIN_BUF_THRESH_L2H</code>	This refers to a low-to-high transition of the input buffer threshold flag.
<code>DSI_WAIT_GSC_AUTOCAL_DONE</code>	This refers to Autocalibration completion
<code>DSI_WAIT_GSC_CHAN_READY</code>	This refers to assertion of the Channel Ready status.
<code>DSI_WAIT_GSC_INIT_DONE</code>	This refers to initialization completion.

4.7.53.4. 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
DSI_WAIT_IO_RX_ABORT	This refers to read requests which have been aborted.
DSI_WAIT_IO_RX_DONE	This refers to read requests which have been satisfied.
DSI_WAIT_IO_RX_ERROR	This refers to read requests which end due to an error.
DSI_WAIT_IO_RX_TIMEOUT	This refers to read requests which end due to the timeout period lapse.

4.7.54. DSI_IOCTL_WAIT_STATUS

This service counts all threads blocked via the `DSI_IOCTL_WAIT_EVENT` IOCTL service (section 4.7.53, page 45), 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	DSI_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 <code>GSC_WAIT_MAIN_*</code> events whose wait requests are to be counted. Refer to section 4.7.53.2 on page 46.
gsc	This specifies the set of <code>DSI_WAIT_GSC_*</code> events whose wait requests are to be counted. Refer to section 4.7.53.3 on page 46.
alt	This is unused by the 24DSI driver and should be zero.
io	This specifies the set of <code>DSI_WAIT_IO_*</code> events whose wait requests are to be counted. Refer to section 4.7.53.4 on page 47.
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.

4.7.55. DSI_IOCTL_XCVR_TYPE

This service selects TTL or LVDS signaling on the external sync and clock cable signals.

Usage

Argument	Description
request	DSI_IOCTL_XCVR_TYPE
arg	s32*

Valid argument values are as follows.

Value	Description
-1	Retrieve the current setting.
DSI_XCVR_TYPE_LVDS	Use LVDS signaling.
DSI_XCVR_TYPE_TTL	Use TTL signaling.

5. The Driver

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

5.1. Files

The device driver files are summarized in the table below.

Description	Files	Location
Source Files	*.c, *.h/driver/
Header File	24dsi.h	
Driver File	24dsi.ko † 24dsi.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 24DSI 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 `24dsi` 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/24dsi.*
```

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/24dsi/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 `rxwxr-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/24dsi` 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/24dsi
```

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/24dsi` 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 24dsi
```

2. Verify that the driver module has been unloaded by issuing the below command. The module name `24dsi` 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	24dsi_dsl.h	.../include/
Library File	24dsi_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 `dsi_open()` there is the utility file `open.c` containing the utility function `dsi_open_util()`. The naming pattern is as follows: API function `dsi_xxxx()`, utility file name `xxxx.c`, utility function `dsi_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 `DSI_IOCTL_QUERY` there is the utility file `util_query.c` containing the utility function `dsi_query()`. The naming pattern is as follows: IOCTL code `DSI_IOCTL_XXXX`, utility file name `util_xxxx.c`, utility function `dsi_xxxx()`.

7.1. Files

The utility files are summarized in the table below.

Description	Files	Location
Source Files	*.c, *.h/utils/
Header File	24dsi_utils.h	.../include/
Library Files	24dsi_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 24DSI. 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	<i>id</i>	.../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
<i>fd</i>	This is the file descriptor used to access the device.
<i>detail</i>	If non-zero the register dump will include details of each register field.

Description	File/Name	Location
Function	<i>dsi_reg_list()</i>	Source File
Source File	<i>util_reg.c</i>	.../utils/
Header File	<i>24dsi_utils.h</i>	.../include/
Library File	<i>24dsi_utils.a</i>	.../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	<i>dsi_config_ai()</i>	Source File
Source File	<i>util_config_ai.c</i>	.../utils/
Header File	<i>24dsi_utils.h</i>	.../include/
Library File	<i>24dsi_utils.a</i>	.../lib/

8.3. Data Transfer Modes

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

8.3.1. PIO - Programmed I/O

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

8.3.2. BMDMA - Block Mode DMA

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

8.3.3. DMDMA - Demand Mode DMA

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

8.4. Multi-Board Synchronization

Multi-board synchronization is a feature of the 24DSI 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. The programming for these two basic options is illustrated via the Multi-Board Sync sample application of section 9.12 on page 60.

8.4.1. Star Configuration

The *star* configuration generally permits all boards in the setup to operate with the least possible phase shift from one board to the next. This is accomplished by configuring all the boards in an identical manner and by wiring the clock and sync signals so that they follow as identical a path as possible from the initiator's output to the input of the initiator and the targets. If there are three or more boards in the setup, then the clock and sync signal must go directly from the initiator's output to a Clock Driver board, as illustrated in Figure 2. If there are only two boards in the setup, then a Clock Driver board is not needed, as illustrated in Figure 3. The table below shows the board programming that is specific to the *star* configuration. See the Multi- Board Sync sample application source code for additional programming requirements (section 9.12, page 60).

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

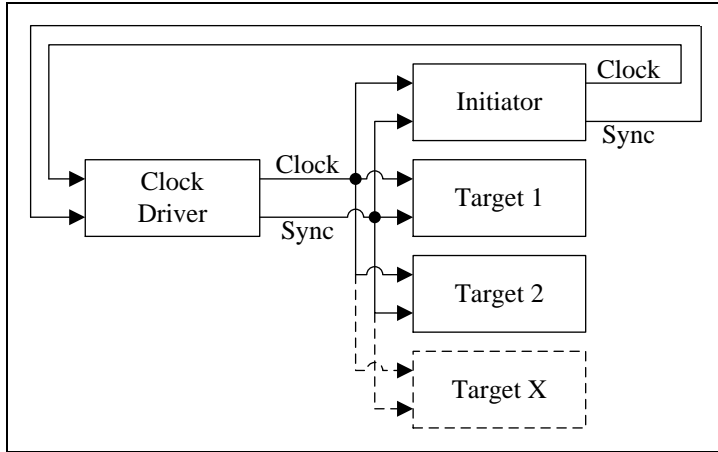


Figure 2 The *star* configuration with three or more boards requires a Clock Driver board.

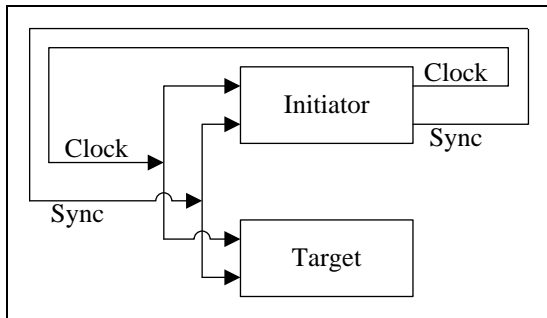


Figure 3 The *star* configuration with only two boards does not require a Clock Driver board.

8.4.2. Daisy Chain Configuration

The *daisy chain* configuration generally permits the most flexible placement of boards and wiring, and does not require a Clock Driver board. This is accomplished by configuring the boards and the wiring so that the clock and sync signals go from the initiator to the first target, then sequentially from the first target to the second and so on. This setup is applicable for any number of boards, as illustrated in Figure 4. The table below shows the board programming that is specific to the *daisy chain* configuration. See the Multi-Board Sync sample application source code for additional programming requirements (section 9.12, page 60).

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

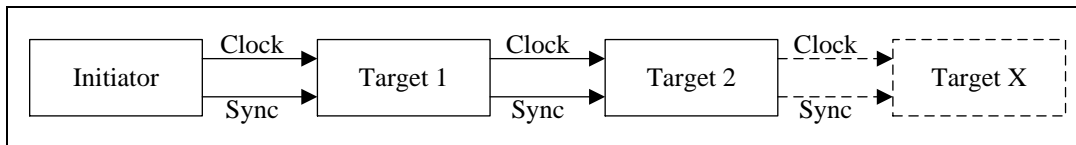


Figure 4 In this configuration the clock and sync signals are daisy chained from one board to the next.

8.5. Clearing the Input Buffer

The subsections below address a few of the basic options for input buffer clearing.

8.5.1. Clear Immediately

One method of clearing the buffer is to use the Clear Buffer IOCTL service (see `DSI_IOCTL_AIN_BUF_CLEAR` in section 4.7.1 on page 24). This service immediately clears the input buffer of any content. However, since this feature is not timed to occur at a scan boundary, it can result in the input buffer containing a partial scan. This method is typically applicable when the input does not need to be cleared at a scan boundary and when only a single board is to be affected.

8.5.2. Clear At a Scan Boundary

Another method of clearing the input buffer is to request that it occur at a scan boundary. This method uses a number of services together. First, configure the board as an Initiator (see `DSI_IOCTL_INIT_MODE` in section 4.7.31 on page 35). Second, configure the board to clear the buffer when there is a Software Sync pulse (see `DSI_IOCTL_SW_SYNC_MODE` in section 4.7.49 on page 43). Finally, to clear the buffer, initiate a Software Sync pulse (see `DSI_IOCTL_SW_SYNC` in section 4.7.48 on page 43). After initiating the Software Sync pulse wait for at least five milliseconds for the operation to complete. This method of clearing the input buffer is applicable either when using a single board, to simplify some data stream processing, or when multiple boards are configured for synchronized operation. (For multi-board synchronization refer to section 8.4 on page 56.)

9. Sample Applications

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

9.1. 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. fref - FREF measurement tool - .../fref/

This application routes the board’s FREF signal to the cable interface for display and measurement.

9.3. fsamp - Sample Rate - .../fsamp/

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

9.4. id - Identify Board - .../id/

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

9.5. regs - Register Access - .../regs/

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

9.6. rxrate - Receive Rate - .../rxrate/

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

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

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

9.8. sbtest - Single Board Test - .../sbtest/

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

9.9. signals - Digital Signals - .../signals/

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

9.10. stream - Stream Rx Data to Disk - .../stream/

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

9.11. sw_sync - Software Sync - .../sw_sync/

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

9.12. mbsync – Multi-Board Sync - .../mbsync/

This application synchronizes multiple boards (2 to 8) for simultaneous input.

NOTE: This application measures the phase difference between the channel zero signals on each board. This is merely a crude approximation. The boards' clock and sync signals must be connected in either a *star* configuration or a *daisy chain* configuration. Also, refer to section 8.4.1 on page 56 for the *star* configuration and section 8.4.2 on page 57 for the *daisy chain* configuration.

NOTE: The signal fed to the channel zero signals on each board must be a sinusoidal wave with a frequency near 5000 Hz (from 4750 Hz to 5250 Hz) with an amplitude from 9.0V to 9.5V (18V to 19V peak-to-peak). If any board is a Low Power version 24DSI, then the amplitude must be from 4.5V to 4.9V (9V to 9.8V peak-to-peak).

Document History

Revision	Description
November 29, 2023	Updated to release version 4.17.107.50.0. Numerous, minor editorial changes. Updated the kernel support table. Updated the description of the Input Buffer Clear service. Updated the description of the Autocalibration service. All Auto_Cal content has been renamed to Autocal.
March 24, 2023	Updated to release version 4.16.103.46.0. Updated the kernel support table. Minor editorial changes.
January 9, 2023	Updated to release version 4.15.101.44.0. Minor editorial changes.
November 21, 2022	Updated to release version 4.14.101.44.0. Updated the information for the open and close calls. Minor editorial updates.
April 28, 2022	Updated to release version 4.13.98.39.2. Minor editorial updates. Updated the kernel support table.
April 25, 2022	Updated to release version 4.13.98.39.1. Minor editorial updates.
April 25, 2022	Updated to release version 4.13.98.39.0. Updated the kernel support table.
October 1, 2021	Updated to release version 4.12.94.37.0. Updated the kernel support table.
August 13, 2021	Updated to release version 4.11.94.36.0. Added section on environment variables. Minor typographic corrections. Added the <code>stream</code> sample application.
May 19, 2021	Updated to release version 4.11.93.36.0. Expanded automatic startup information.
June 8, 2020	Updated to release version 4.11.91.32.0. Updated the kernel support table. Minor editorial changes. Added <code>WAIT_EVENT</code> note. Expanded automatic startup information.
July 15, 2019	Updated to release version 4.10.86.28.0. Updated the kernel support table. Minor editorial change. Added a licensing subsection.
April 25, 2019	Updated to release version 4.9.85.27.0.
April 22, 2019	Updated to release version 4.8.85.27.0. Minor editorial changes.
December 14, 2018	Updated to release version 4.8.81.26.1. Minor editorial changes. Added support for the 24DSI8R.
November 21, 2018	Updated to release version 4.8.81.26.0. Minor editorial changes.
November 5, 2018	Updated to release version 4.7.81.26.0. Minor editorial changes. Minor editorial changes. Updated the inside cover page. Updated the CPU and kernel support section. Updated Block Mode DMA macro and associated information. Minor document overhaul.
December 26, 2017	Updated to release version 4.6.73.20.1. Minor editorial changes.
December 15, 2017	Updated to release version 4.6.73.20.0.
December 5, 2017	Updated to release version 4.5.73.20.1. Added <code>dsi_init()</code> documentation.
September 28, 2017	Updated to release version 4.5.73.20.0. Document, interface and directory restructured.
December 5, 2016	Updated to release version 4.4.68.18.0. Removed the <code>built</code> field from the <code>/proc/</code> file. Updated the kernel support table. Updated the command line arguments for the <code>FSAMP</code> and <code>rxrate</code> sample applications. Organized sample applications alphabetically. Updated the usage of the Wait Event <code>timeout_ms</code> field. Updated material on the open call. Added open access mode descriptions. Updated the kernel support table. Added support for infinite I/O timeouts. Updated the operating information section. Made various miscellaneous updates. Some document reorganization.
March 3, 2016	Updated to version 4.3.65.13.0.
March 3, 2016	Updated to version 4.2.65.13.0. Added <code>savedata</code> command line arguments <code>-pio</code> , <code>-dma</code> , and <code>-dmdma</code> .
September 25, 2015	Updated to version 4.1.61.9.0.
September 15, 2015	Updated to version 4.0.60.8.0.
July 31, 2015	Updated to version 4.0.59.7.0. Updated the device node name to include a period before the device index. Removed double underscore that prefaced various data types.
November 26, 2014	Updated to version 3.18.57.0.
February 26, 2014	Updated to version 3.17.52.0. Updated the kernel support data.
January 8, 2014	Updated to version 3.16.51.0. Updated the kernel support data.

November 13, 2013	Updated to version 3.16.49.0.
October 5, 2013	Updated to version 3.15.48.0.
July 16, 2013	Updated to version 3.14.45.0. Updated the kernel support data.
August 21, 2012	Updated to version 3.14.39.0. Added the <code>DSI_IOCTL_CHANNELS_READY_IOCTL</code> service. Added the <code>mbsync</code> sample application. Removed the <code>2bsync</code> sample application.
July 20, 2012	Updated to version 3.13.39.0.
July 13, 2012	Updated to version 3.12.39.0. Updated the kernel support data.
March 20, 2012	Updated to version 3.11.36.0.
February 16, 2012	Updated to version 3.10.36.0. Added the <code>DSI_QUERY_D23_INV_EXT_TRIG</code> and <code>DSI_QUERY_REG_ICMR</code> query options. Added the <code>DSI_GSC_ICMR</code> register. Added the <code>DSI_IOCTL_TA_INV_EXT_TRIGGER</code> , <code>DSI_IOCTL_COUPLING_MODE_AC</code> and <code>DSI_IOCTL_COUPLING_MODE_DC_IOCTL</code> services.
December 22, 2011	Updated to version 3.9.34.0.
November 14, 2011	Updated to version 3.8.32.0.
October 12, 2011	Updated to version 3.8.29.0. Various editorial changes. Updated the CPU and Kernel Support information. Updated the comments for the Initialize IOCTL service. Changed the spelling of various Autocalibration related software items. Added the signals sample application.
June 4, 2010	Updated to version 3.7.16.0. Added the <code>IRQ_SEL</code> and <code>WAIT_IOCTL</code> services.
December 28, 2009	Updated to version 3.6.13.0.
June 23, 2009	Updated to version 3.5.8.0.
May 28, 2009	Updated to version 3.4.7.0.
April 22, 2009	Updated to version 3.3.5.0.
April 17, 2009	Updated to version 3.2.5.0.
April 16, 2009	Updated to version 3.2.4.0.
February 27, 2009	Updated to version 3.1.2.0. Added support for the 24DSI6LN boards.
December 11, 2008	Initial release.