

24DSI

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

**PCI/PMC/CPCI/PC104P-...
...-24DSI32/12/6**

Linux Device Driver User Manual

**Manual Revision: December 22, 2011
Driver Release Version 3.9.34.0**

**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 © 2008-2011, **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 material, 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 to ECO control, **General Standards Corporation** assumes no responsibility for any errors that may exist in this document. No commitment is made to update or keep current the information contained in this document.

General Standards Corporation does not assume any liability arising out of the application or use of any 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 product to improve reliability, performance, function, or design.

ALL RIGHTS RESERVED.

The Purchaser of this software may use or modify in source form the subject software, but not to re-market or distribute it to outside agencies or separate internal company divisions. The software, however, may be embedded in the Purchaser's distributed software. In the event the Purchaser's customers require the software source code, then they would have to purchase their own copy of the software.

General Standards Corporation makes no warranty of any kind with regard to this software, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose and makes this software available solely on an "as-is" basis. **General Standards Corporation** reserves the right to make changes in this software without reservation and without notification to its users.

The information in this document is subject to change without notice. This document may be copied or reproduced provided it is in support of products from **General Standards Corporation**. For any other use, no part of this document may be copied or reproduced in any form or by any means without prior written consent of **General Standards Corporation**.

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

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

Table of Contents

1. Introduction.....	7
1.1. Purpose.....	7
1.2. Acronyms.....	7
1.3. Definitions	7
1.4. Software Overview	7
1.5. Hardware Overview	7
1.6. Reference Material.....	8
2. Installation.....	9
2.1. CPU and Kernel Support.....	9
2.1.1. 32-bit Support Under 64-bit Environments	9
2.2. The /proc File System	9
2.3. File List.....	10
2.4. Directory Structure.....	10
2.5. Installation	10
2.6. Removal.....	11
2.7. Overall Make Script.....	11
3. The Driver.....	12
3.1. Build	12
3.2. Startup.....	12
3.2.1. Manual Driver Startup Procedures	12
3.2.2. Automatic Driver Startup Procedures.....	13
3.3. Verification	13
3.4. Version.....	13
3.5. Shutdown	14
4. Document Source Code Examples.....	15
4.1. Build	15
4.2. Library Use	15
5. Sample Applications	16
5.1. id - Identify Board.....	16
5.1.1. Build	16
5.1.2. Execute	16
5.2. regs - Register Access	17
5.2.1. Build	17
5.2.2. Execute	17
5.3. sbtest - Single Board Test	18
5.3.1. Build	18

5.3.2. Execute	18
5.4. billion - Billion Sample Read.....	19
5.4.1. Build	19
5.4.2. Execute	19
5.5. rxrate - Receive Rate.....	20
5.5.1. Build	20
5.5.2. Execute	20
5.6. fsamp - FSAMP Calculator	21
5.6.1. Build	21
5.6.2. Execute	21
5.7. savedata - Save Acquired Data	22
5.7.1. Build	22
5.7.2. Execute	22
5.8. sw_sync - Software Sync	23
5.8.1. Build	23
5.8.2. Execute	23
5.9. 2bsync - Two Board Sync	24
5.9.1. Build	24
5.9.2. Execute	24
5.10. signals - Digital Signals	26
5.10.1. Build	26
5.10.2. Execute	26
6. Driver Interface.....	27
6.1. Macros	27
6.1.1. IOCTL	27
6.1.2. Registers	27
6.2. Functions.....	28
6.2.1. close()	28
6.2.2. ioctl()	28
6.2.3. open().....	29
6.2.4. read()	30
6.3. IOCTL Services	31
6.3.1. DSI_IOCTL_ADC_RATE_OUT	31
6.3.2. DSI_IOCTL_AIN_BUF_CLEAR	32
6.3.3. DSI_IOCTL_AIN_BUF_INPUT	32
6.3.4. DSI_IOCTL_AIN_BUF_OVERFLOW	32
6.3.5. DSI_IOCTL_AIN_BUF_THRESH.....	33
6.3.6. DSI_IOCTL_AIN_BUF_UNDERFLOW	33
6.3.7. DSI_IOCTL_AIN_COUPLING.....	33
6.3.8. DSI_IOCTL_AIN_FILTER	34
6.3.9. DSI_IOCTL_AIN_FILTER_XX_XX.....	34
6.3.10. DSI_IOCTL_AIN_FILTER_SEL	35
6.3.11. DSI_IOCTL_AIN_MODE	35
6.3.12. DSI_IOCTL_AIN_RANGE	35
6.3.13. DSI_IOCTL_AIN_RANGE_XX_XX	36
6.3.14. DSI_IOCTL_AIN_RANGE_SEL	37
6.3.15. DSI_IOCTL_AUTO_CALIBRATE.....	37
6.3.16. DSI_IOCTL_CH_GRP_X_SRC.....	37
6.3.17. DSI_IOCTL_CHANNEL_ORDER	38

6.3.18. DSI_IOCTL_DATA_FORMAT	38
6.3.19. DSI_IOCTL_DATA_WIDTH.....	38
6.3.20. DSI_IOCTL_EXT_CLK_SRC.....	39
6.3.21. DSI_IOCTL_EXT_TRIG.....	39
6.3.22. DSI_IOCTL_GPS_ENABLE.....	39
6.3.23. DSI_IOCTL_GPS_LOCKED	40
6.3.24. DSI_IOCTL_GPS_POLARITY	40
6.3.25. DSI_IOCTL_GPS_RATE_LOCKED	40
6.3.26. DSI_IOCTL_GPS_TARGET_RATE.....	41
6.3.27. DSI_IOCTL_GPS_TOLERANCE	41
6.3.28. DSI_IOCTL_INIT_MODE	41
6.3.29. DSI_IOCTL_INITIALIZE	42
6.3.30. DSI_IOCTL_IRQ_SEL.....	42
6.3.31. DSI_IOCTL_PLL_REF_FREQ_HZ.....	42
6.3.32. DSI_IOCTL_QUERY	42
6.3.33. DSI_IOCTL_RATE_DIV_X_NDIV	45
6.3.34. DSI_IOCTL_RATE_GEN_X_NRATE.....	45
6.3.35. DSI_IOCTL_RATE_GEN_X_NREF.....	45
6.3.36. DSI_IOCTL_RATE_GEN_X_NVCO.....	46
6.3.37. DSI_IOCTL_REG_MOD.....	46
6.3.38. DSI_IOCTL_REG_READ	46
6.3.39. DSI_IOCTL_REG_WRITE	47
6.3.40. DSI_IOCTL_RX_IO_ABORT.....	47
6.3.41. DSI_IOCTL_RX_IO_MODE.....	48
6.3.42. DSI_IOCTL_RX_IO_OVERFLOW	48
6.3.43. DSI_IOCTL_RX_IO_TIMEOUT	48
6.3.44. DSI_IOCTL_RX_IO_UNDERFLOW	49
6.3.45. DSI_IOCTL_SW_SYNC	49
6.3.46. DSI_IOCTL_SW_SYNC_MODE.....	49
6.3.47. DSI_IOCTL_THRES_FLAG_CBL	50
6.3.48. DSI_IOCTL_WAIT_CANCEL.....	50
6.3.49. DSI_IOCTL_WAIT_EVENT.....	51
6.3.50. DSI_IOCTL_WAIT_STATUS	53
6.3.51. DSI_IOCTL_XCVR_TYPE.....	53
7. Operation.....	55
7.1. Data Reception.....	55
7.2. Multi-Board Synchronization	55
7.2.1. Star Configuration	55
7.2.2. Daisy Chain Configuration.....	56
7.3. Clearing the Input Buffer.....	57
7.3.1. Clear Immediately	57
7.3.2. Clear At a Scan Boundary	57
Document History	58

Table of Figures

Figure 1 The <i>star</i> configuration with three or more boards requires a Clock Driver board.....	56
Figure 2 The <i>star</i> configuration with only two boards does not require a Clock Driver board.	56
Figure 3 In this configuration the clock and sync signals are daisy chained from one board to the next.	57

1. Introduction

This user manual applies to driver release version 3.9.34.0.

NOTE: This driver is designed to function with both PLL and non-PLL versions of the 24DSI board family.

1.1. Purpose

The purpose of this document is to describe the interface to the 24DSI Linux device driver. This driver software provides the interface between "Application Software" and a 24DSI board. The interface to the board is at the device level.

1.2. Acronyms

The following is a list of commonly occurring acronyms used throughout this document.

Acronyms	Description
CPCI	Compact PCI
DMA	Direct Memory Access
GSC	General Standards Corporation
PC104+	This refers to the PC104+ form factor of PCI boards.
PCI	Peripheral Component Interconnect
PMC	PCI Mezzanine Card

1.3. Definitions

The following is a list of commonly occurring terms used throughout this document.

Term	Definition
24DSI	This is used as a general reference to any board supported by this driver.
Application	Application means the user mode process, which runs in the user space with user mode privileges.
Driver	Driver means the kernel mode device driver, which runs in the kernel space with kernel mode privileges.

1.4. Software Overview

The 24DSI driver software executes under control of the Linux operating system and runs in Kernel Mode as a Kernel Mode device driver. The 24DSI device driver is implemented as a standard dynamically loadable Linux device driver written in the C programming language. With the driver, user applications are able to open and close a device and, while open, perform read and I/O control operations. Write operations to the board are not supported.

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 auto-calibration capability.

1.6. Reference Material

The following reference material may be of particular benefit in using a 24DSI board and this driver. 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 *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>

2. Installation

2.1. CPU and Kernel Support

The driver is designed to operate with Linux kernel versions 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	x86	
		32-bit	64-bit
2.6.38	Red Hat Fedora Core 15	Yes	Yes
2.6.35	Red Hat Fedora Core 14	Yes	Yes
2.6.31	Red Hat Fedora Core 13	Yes	Yes
2.6.31	Red Hat Fedora Core 12	Yes	Yes
2.6.19	Red Hat Fedora Core 11	Yes	Yes
2.6.27	Red Hat Fedora Core 10	Yes	Yes
2.6.25	Red Hat Fedora Core 9	Yes	Yes
2.6.23	Red Hat Fedora Core 8	Yes	Yes
2.6.21	Red Hat Fedora Core 7	Yes	Yes
2.6.18	Red Hat Fedora Core 6	Yes	Yes
2.6.15	Red Hat Fedora Core 5	Yes	Yes
2.6.11	Red Hat Fedora Core 4	Yes	Yes
2.6.9	Red Hat Fedora Core 3	Yes	Yes
2.4.21	Red Hat Enterprise Linux Workstation Release 3	Yes	
2.4.7	Red Hat Linux 7.2	Yes	
2.2.14	Red Hat Linux 6.2	Yes	

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

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

NOTE: The driver has not been tested for SMP operation.

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

```
version: 3.9.34
built: Dec 22 2011, 19:42:02
32-bit support: yes
boards: 1
models: 24DSI12
```

Entry	Description
version	This gives the driver version number in the form x.x.x.
built	This gives the driver build date and time as a string. It is given in the C form of <code>printf("%s, %s", __DATE__, __TIME__)</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.

2.3. File List

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

File	Description
24dsi.linux.tar.gz	This archive contains the driver, the samples and all related sources.
24dsi_linux_um.pdf	This is a PDF version of this user manual, which is included in the archive.

2.4. Directory Structure

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

Directory	Content
24dsi	This is the source root directory. The user manual and overall make script are placed here.
24dsi\2bsync	This directory contains the Two Board Sync sample application. Refer to section 5.8 on page 23.
24dsi\billion	This directory contains the Billion Sample Read sample application. Refer to section 0 on page 16.
24dsi\docsrc	This directory contains the Document Source Code Examples. Refer to section 4 on page 15.
24dsi\driver	This directory contains the driver and its sources. Refer to section 3 on page 12.
24dsi\fsamp	This directory contains the Sample Rate sample application. Refer to section 5.6 on page 21.
24dsi\rxrate	This directory contains the Receive Rate sample application. Refer to section 5.5 on page 20.
24dsi\savedata	This directory contains the Save Acquired Data sample application. Refer to section 5.6 on page 21.
24dsi\sbtest	This directory contains the Single Board Test application. Refer to section 5.2.2 on page 17.
24dsi\signals	This directory contains the Signals sample application. Refer to section 5.10 on page 26.
24dsi\utils	This directory contains utility sources used by the sample applications.

2.5. Installation

Install the driver and its related files following the below listed steps. This includes the device driver, the documentation source code, and the sample applications.

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

Follow the below steps to remove the driver and its related files. This includes the device driver, the documentation source code, and the sample applications.

1. Shutdown the driver as described in section 3.5 on page 14.
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 automated startup procedure was adopted (see section 3.2.2 on page 13), 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, and it will also load the driver. The script is named `make_all`. Follow the below steps to perform an overall make and to load the driver.

1. Change to the driver's directory, which may be `/usr/src/linux/drivers/24dsi`.
2. Issue the following command to make all archive targets and to load the driver.

```
./make_all
```

3. The Driver

The driver and its related files are contained in the archive file `24dsi.linux.tar.gz`. The driver's files are summarized in the table below.

File	Description
<code>driver/*.c</code>	The driver source files.
<code>driver/*.h</code>	The driver header files.
<code>driver/start</code>	Shell script to install the driver executable and device nodes.
<code>driver/24dsi.h</code>	This is the main driver header file. This header should be included by 24DSI applications.
<code>driver/Makefile</code>	This is the driver make file.

3.1. Build

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

Follow the below steps to build the driver.

1. Change to the directory where the driver and its sources are installed, which may be `/usr/src/linux/drivers/24dsi/driver`.
2. Remove all existing build targets by issuing the below command.

```
make clean
```

3. Build the driver by issuing the below command.

```
make all
```

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.

3.2. 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 insure that the driver module in the install directory is the module that is loaded. This is accomplished by making sure that an already loaded module is first unloaded before attempting to load the module from the disk drive. In addition, the script also deletes and recreates the device nodes. This is done to insure that the device nodes in use have the same major number as assigned dynamically to the driver by the kernel, and so that the number of device nodes correspond to the number of boards identified by the driver.

3.2.1. Manual Driver Startup Procedures

Start the driver manually by following the below listed steps.

1. Login as root user, as some of the steps require root privileges.
2. Change to the directory where the driver are installed, which may be `/usr/src/linux/drivers/24dsi/driver`.

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

```
./start
```

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

NOTE: The 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.

4. 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
```

5. 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*
```

3.2.2. 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
```

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.

3.3. 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
```

3.4. 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.

3.5. Shutdown

Shutdown the driver following the below listed steps.

1. Login as root user, as some of the steps require root privileges.
2. If the driver is currently loaded then issue the below command to unload the driver.

`rmmod 24dsi`
3. 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`

4. Document Source Code Examples

The archive file `24dsi.linux.tar.gz` contains all of the source code examples included in this document. In addition, the code is built into a statically linkable library usable with 24DSI console applications. The library and sources are delivered undocumented and unsupported. 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. These files are located in the `docsrc` subdirectory under the 24DSI root directory.

File	Description
<code>docsrc/*.c</code>	These are the C source files.
<code>docsrc/24dsi_dsl.h</code>	This is the library header file.
<code>docsrc/makefile</code>	This is the library make file.
<code>docsrc/makefile.dep</code>	This is an automatically generated make dependency file.

4.1. Build

Follow the below steps to compile the example files and build the library.

1. Change to the directory where the documentation sources are installed, which may be `/usr/src/linux/drivers/24dsi/docsrc`.
2. Remove all 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 all
```

4.2. Library Use

The library is used both at application compile time and at application link time. Compile time use has two requirements. First, include the header file `24dsi_dsl.h` in each module referencing a library component. Second, expand the include file search path to search the directory where the library header is located, which may be `/usr/src/linux/drivers/24dsi/docsrc`. Link time use also has two requirements. First, include the static library `24dsi_dsl.a` in the list of files to be linked into the application. Second, expand the library file search path to search the directory where the library is located, which may be `/usr/src/linux/drivers/24dsi/docsrc`.

5. Sample Applications

NOTE: The sample applications are unsupported and are provided without documentation.

5.1. id - Identify Board

This sample console application provides a command line driven Linux application that provides 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. The application's sources are summarized in the below table.

File	Description
id/*.c	These are the application's source files.
id/main.h	This is the application's header file.
id/makefile	This is the application make file.
id/makefile.dep	This is an automatically generated make dependency file.
docsrc/*	These are utility sources used by the application.
utils/*	These are utility sources used by the application.

5.1.1. Build

Follow the below steps to build the sample application.

1. Change to the directory where the sample application sources are installed (.../id).

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

```
make clean
```

3. Build the application by issuing the below command.

```
make all
```

5.1.2. Execute

Follow the below steps to execute the sample application.

1. Change to the directory where the sample application sources are installed (.../id).

2. Start the sample application by issuing the command given below. Once started the application will automatically output identification information. A single iteration should take less than a second. The command line arguments are described in the table below.

```
./id <index>
```

Argument	Description
index	This is the zero based index of the board to access.

5.2. regs - Register Access

This sample console application provides a menu based command line Linux application that permits interactive access to the board's registers, including write access to the GSC specific registers. The application's sources are summarized in the below table.

File	Description
regs/*.c	These are the application's source files.
regs/main.h	This is the application's header file.
regs/makefile	This is the application make file.
regs/makefile.dep	This is an automatically generated make dependency file.
docsrc/*	These are utility sources used by the application.
utils/*	These are utility sources used by the application.

5.2.1. Build

Follow the below steps to build the sample application.

1. Change to the directory where the sample application sources are installed (.../regs).
2. Remove all existing build targets by issuing the below command.

```
make clean
```

3. Build the application by issuing the below command.

```
make all
```

5.2.2. Execute

Follow the below steps to execute the sample application.

1. Change to the directory where the sample application sources are installed (.../regs).
2. Start the sample application by issuing the command given below. The command line argument is described in the table below.

```
./regs <index>
```

Argument	Description
index	This is the zero based index of the board to access.

3. Select the desired options according to the menus presented. Select the main menu exit option when finished.

5.3. sbtest - Single Board Test

This sample console application provides a command line driven Linux application that tests the functionality of the driver and a specified board. The application's sources are summarized in the below table.

File	Description
sbtest/*.c	These are the application's source files.
sbtest/main.h	This is the application's header file.
sbtest/makefile	This is the application make file.
sbtest/makefile.dep	This is an automatically generated make dependency file.
docsrc/*	These are utility sources used by the application.
utils/*	These are utility sources used by the application.

5.3.1. Build

Follow the below steps to build the sample application.

1. Change to the directory where the sample application sources are installed (.../sbtest).

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

```
make clean
```

3. Build the application by issuing the below command.

```
make all
```

5.3.2. Execute

NOTE: This application should be run with no cable attached.

Follow the below steps to execute the sample application.

1. Change to the directory where the sample application sources are installed (.../sbtest).

2. Start the sample application by issuing the command given below. Once started the application will automatically performs a series of test operations. A single iteration should take less than 10 minutes to complete. The command line arguments are described in the table below.

```
./sbtest <-c> <-C> <-m#> <-n#> <index>
```

Argument	Description
-c	Repeat the operation until an error is encountered.
-C	Repeat the operation, but continue even if errors are encountered.
-m#	When repeating the operation, stop after “#” minutes, where “#” is a decimal number.
-n#	When repeating the operation, stop after “#” iterations, where “#” is a decimal number.
index	This is the zero based index of the board to access.

5.4. billion - Billion Sample Read

This sample console application provides a command line driven Linux application that will configure the designated board then read in a billion samples. The data is discarded after it is read. The application's sources are summarized in the below table.

File	Description
billion/*.c	These are the application's source files.
billion/main.h	This is the application's header file.
billion/makefile	This is the application make file.
billion/makefile.dep	This is an automatically generated make dependency file.
docsrc/*	These are utility sources used by the application.
utils/*	These are utility sources used by the application.

5.4.1. Build

Follow the below steps to build the sample application.

1. Change to the directory where the sample application sources are installed (.../billion).
2. Remove all existing build targets by issuing the below command.

```
make clean
```

3. Build the application by issuing the below command.

```
make all
```

5.4.2. Execute

Follow the below steps to execute the sample application.

1. Change to the directory where the sample application sources are installed (.../billion).
2. Start the sample application by issuing the command given below. Once started the application will configure the board and begin reading data. With the default options a single iteration may take from 10 to 30 minutes to complete. The command line arguments are described in the table below.

```
./billion <-c> <-C> <-m#> <-n#> <index>
```

Argument	Description
-c	Repeat the operation until an error is encountered.
-C	Repeat the operation, but continue even if errors are encountered.
-m#	When repeating the operation, stop after “#” minutes, where “#” is a decimal number.
-n#	When repeating the operation, stop after “#” iterations, where “#” is a decimal number.
index	This is the zero based index of the board to access.

5.5. rxrate - Receive Rate

This sample console application provides a command line driven Linux application that will read a specified amount of data from a specified board. The data is discarded after it is read. The application's sources are summarized in the below table.

File	Description
rxrate/*.c	These are the application's source files.
rxrate/main.h	This is the application's header file.
rxrate/makefile	This is the application make file.
rxrate/makefile.dep	This is an automatically generated make dependency file.
docsrc/*	These are utility sources used by the application.
utils/*	These are utility sources used by the application.

5.5.1. Build

Follow the below steps to build the sample application.

1. Change to the directory where the sample application sources are installed (.../rxrate).
2. Remove all existing build targets by issuing the below command.

```
make clean
```

3. Build the application by issuing the below command.

```
make all
```

5.5.2. Execute

Follow the below steps to execute the sample application.

1. Change to the directory where the sample application sources are installed (.../rxrate).
2. Start the sample application by issuing the command given below. Once started the application will automatically perform a series of operations. With the default options a single iteration should take less than 30 seconds to complete. The command line arguments are described in the table below.

```
./rxrate <-c> <-C> <-m#> <-n#> <-r#> <index>
```

Argument	Description
-c	Repeat the operation until an error is encountered.
-C	Repeat the operation, but continue even if errors are encountered.
-m#	When repeating the operation, stop after “#” minutes, where “#” is a decimal number.
-n#	When repeating the operation, stop after “#” iterations, where “#” is a decimal number.
-r#	This specifies the number of megabytes to read where “#” is a decimal number.
index	This is the zero based index of the board to access.

5.6. fsamp - FSAMP Calculator

This sample console application computes the optimum theoretical parameters that can be used to produce a user specified sample rate. The application's sources are summarized in the below table.

File	Description
fsamp/*.c	These are the application's source files.
fsamp/main.h	This is the application's header file.
fsamp/makefile	This is the application make file.
fsamp/makefile.dep	This is an automatically generated make dependency file.
docsrc/*	These are utility sources used by the application.
utils/*	These are utility sources used by the application.

5.6.1. Build

Follow the below steps to build the sample application.

1. Change to the directory where the sample application sources are installed (.../fsamp).
2. Remove all existing build targets by issuing the below command.

```
make clean
```

3. Build the application by issuing the below command.

```
make all
```

5.6.2. Execute

Follow the below steps to execute the sample application.

1. Change to the directory where the sample application sources are installed (.../fsamp).
2. Start the sample application by issuing the command given below. Once started the application automatically performs its operations. A single iteration should take only a few seconds to complete. The command line arguments are described in the table below.

```
./fsamp <-o#> <-r#> <-s#> <index>
```

Argument	Description
-o#	This requests that the application report all configuration options that produce a sample rate within # number of samples/second of the requested rate.
-r#	This specifies a custom value to use for the reference clock frequency, Fref.
-s#	This specifies the desired sample rate, Fsamp.
index	This is the zero based index of the board to access.

5.7. savedata - Save Acquired Data

This sample console application provides a command line driven Linux application that configures a selected board, reads about one million samples, then saves the data to a text file. The saved data's file name is `data.txt` and includes one line for each scan. The application's sources are summarized in the below table.

File	Description
<code>savedata/*.c</code>	These are the application's source files.
<code>savedata/main.h</code>	This is the application's header file.
<code>savedata/makefile</code>	This is the application make file.
<code>savedata/makefile.dep</code>	This is an automatically generated make dependency file.
<code>docsrc/*</code>	These are utility sources used by the application.
<code>utils/*</code>	These are utility sources used by the application.

5.7.1. Build

Follow the below steps to build the sample application.

1. Change to the directory where the sample application sources are installed (`.../savedata`).
2. Remove all existing build targets by issuing the below command.

```
make clean
```

3. Build the application by issuing the below command.

```
make all
```

5.7.2. Execute

Follow the below steps to execute the sample application.

1. Change to the directory where the sample application sources are installed (`.../savedata`).
2. Start the sample application by issuing the command given below. Once started the application automatically performs its operations. With the default options a single iteration should take less than 40 seconds to complete. The command line arguments are described in the table below.

```
./savedata <-c> <-C> <-m#> <-n#> <index>
```

Argument	Description
<code>-c</code>	Repeat the operation until an error is encountered.
<code>-C</code>	Repeat the operation, but continue even if errors are encountered.
<code>-m#</code>	When repeating the operation, stop after “#” minutes, where “#” is a decimal number.
<code>-n#</code>	When repeating the operation, stop after “#” iterations, where “#” is a decimal number.
<code>index</code>	This is the zero based index of the board to access.

5.8. sw_sync - Software Sync

This sample console application provides a command line driven Linux application that repetitively generates a sync pulse on the cable's Software Sync output signal. The purpose of this application is to permit verification of the signal's operation and presence, and to more easily permit test equipment to be configured to capture the signal. The application's sources are summarized in the below table.

File	Description
sw_sync/*.c	These are the application's source files.
sw_sync/main.h	This is the application's header file.
sw_sync/makefile	This is the application make file.
sw_sync/makefile.dep	This is an automatically generated make dependency file.
docsrc/*	These are utility sources used by the application.
utils/*	These are utility sources used by the application.

5.8.1. Build

Follow the below steps to build the sample application.

1. Change to the directory where the sample application sources are installed (.../sw_sync).
2. Remove all existing build targets by issuing the below command.

```
make clean
```

3. Build the application by issuing the below command.

```
make all
```

5.8.2. Execute

NOTE: This application should be run with no cable attached.

Follow the below steps to execute the sample application.

1. Change to the directory where the sample application sources are installed (.../sw_sync).
2. Start the sample application by issuing the command given below. Once started the application will generate a series of cable sync pulses. A single iteration should take no more than a second or so. The command line arguments are described in the table below.

```
./sw_sync <-c> <-C> <-m#> <-n#> <index>
```

Argument	Description
-c	Repeat the operation until an error is encountered.
-C	Repeat the operation, but continue even if errors are encountered.
-m#	When repeating the operation, stop after “#” minutes, where “#” is a decimal number.
-n#	When repeating the operation, stop after “#” iterations, where “#” is a decimal number.
index	This is the zero based index of the board to access.

5.9. 2bsync - Two Board Sync

This sample console application provides a command line driven Linux application that synchronizes two boards. This application's primary purpose is to demonstrate the steps needed for multi-board synchronization (refer to `sync.c`). To do this, the clock and sync signals must be connected in either a *star* configuration or a *daisy chain* configuration. (Refer to section 7.2.1 on page 55 for the *star* configuration and section 7.2.2 on page 56 for the *daisy chain* configuration. The application's sources are summarized in the below table.

File	Description
2bsync/*.c	These are the application's source files.
2bsync/main.h	This is the application's header file.
2bsync/makefile	This is the application make file.
2bsync/makefile.dep	This is an automatically generated make dependency file.
docsrc/*	These are utility sources used by the application.
utils/*	These are utility sources used by the application.

5.9.1. Build

Follow the below steps to build the sample application.

1. Change to the directory where the sample application sources are installed (`.../2bsync`).
2. Remove all existing build targets by issuing the below command.

```
make clean
```

3. Build the application by issuing the below command.

```
make all
```

5.9.2. Execute

NOTE: This application measures the phase difference between the channel zero signals on two boards. The boards' clock and sync signals must be connected in either a *star* configuration or a *daisy chain* configuration. See below for additional information. Also see section 7.2.1 on page 55 for the *star* configuration and section 7.2.2 on page 56 for the *daisy chain* configuration.

NOTE: The signal fed to the channel zero signals on both boards must be a sinusoidal wave with a frequency near 5000 Hz (from 4750 Hz to 5250 Hz) with an amplitude from 4.5V to 9.5V (9V peak-to-peak and 19V peak-to-peak, respectively). If either 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).

Follow the below steps to execute the sample application. The configuration that connects the clock and sync signals, *star* or *daisy chain*, is specified using command line arguments. For additional information refer to section 7.2.1 on page 55 for the *star* configuration and section 7.2.2 on page 56 for the *daisy chain* configuration.

1. Change to the directory where the sample application sources are installed (`.../2bsync`).
2. Start the sample application by issuing the command given below. Once started the application automatically configures and synchronizes the two boards. A single iteration should take less than 45 seconds to complete. Each iteration will take longer if the Repeat Validation option (`-rv#`) is specified. The command line arguments are described in the table below.

```
./2bsync <-c> <-C> <-d> <-m#> <-n#> <-rv#> <-s> <initiator> <target>
```


Argument	Description
-c	Repeat the operation until an error is encountered.
-C	Repeat the operation, but continue even if errors are encountered.
-d	This specifies that the clock and sync cable signals are connected in a <i>daisy chain</i> configuration. Refer to section 7.2.2 on page 56.
-m#	When repeating the operation, stop after “#” minutes, where “#” is a decimal number.
-n#	When repeating the operation, stop after “#” iterations, where “#” is a decimal number.
-rv#	This specifies that additional data be read from the boards and that the synchronization validation be performed on this additional data. The number “#” is the number of additional times the read and validation are to be repeated.
-s	This specifies that the clock and sync signals are connected in a <i>star</i> configuration. Refer to section 7.2.1 on page 55.
initiator	This is the zero based index of the board designated as the initiator.
target	This is the zero based index of the board designated as the target.

5.10. signals - Digital Signals

This sample console application provides a command line driven Linux application that configures the board to output the clock and sync signals on the cable interface. The purpose of this application is to permit verification of the signal's operation and presence, and to more easily permit test equipment to be configured to capture the signals. The output clock should be near about 30MHz. The output sync signal should produce a 2.5 μ s pulse somewhere from 50K to 300K times per second. The application's sources are summarized in the below table.

File	Description
signals/*.c	These are the application's source files.
signals/main.h	This is the application's header file.
signals/makefile	This is the application make file.
signals/makefile.dep	This is an automatically generated make dependency file.
docsrc/*	These are utility sources used by the application.
utils/*	These are utility sources used by the application.

5.10.1. Build

Follow the below steps to build the sample application.

1. Change to the directory where the sample application sources are installed (.../signals).
2. Remove all existing build targets by issuing the below command.

```
make clean
```

3. Build the application by issuing the below command.

```
make all
```

5.10.2. Execute

NOTE: This application should be run with no cable attached.

Follow the below steps to execute the sample application.

1. Change to the directory where the sample application sources are installed (.../signals).
2. Start the sample application by issuing the command given below. Once started the application will generate a series of cable sync pulses. A single iteration should take about 30 seconds to complete. The command line arguments are described in the table below.

```
./signals <-c> <-C> <-m#> <-n#> <-s#> <index>
```

Argument	Description
-c	Repeat the operation until an error is encountered.
-C	Repeat the operation, but continue even if errors are encountered.
-m#	When repeating the operation, stop after “#” minutes, where “#” is a decimal number.
-n#	When repeating the operation, stop after “#” iterations, where “#” is a decimal number.
-s#	Generate the output signals for this many seconds, where “#” is a decimal number. The default is 30 seconds.
index	This is the zero based index of the board to access.

6. Driver Interface

The 24DSI driver conforms to the device driver standards required by the Linux Operating System and contains the standard driver entry points. The device driver provides a standard driver interface to 24DSI boards for Linux applications. The interface includes various macros, data types and functions, all of which are described in the following paragraphs. The 24DSI specific portion of the driver interface is defined in the header file `24dsi.h`, portions of which are described in this section. The header defines numerous items in addition to those described here.

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

6.1. Macros

The driver interface includes the following macros, which are defined in `24dsi.h`. The header also contains various other utility type macros, which are provided without documentation.

6.1.1. IOCTL

The IOCTL macros are documented in section 6.3 beginning on page 31.

6.1.2. Registers

The following gives the complete set of 24DSI registers.

6.1.2.1. GSC Registers

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

Macro	Description	Information
DSI_GSC_AVR	Autocal Values Register	
DSI_GSC_BCFGR	Board Configuration Register	
DSI_GSC_BCTLR	Board Control Register	
DSI_GSC_BSR	Buffer Size Register	
DSI_GSC_IBCR	Input Buffer Control Register	
DSI_GSC_IDBR	Input Data Buffer Register	
DSI_GSC_GSR	GPS Synchronization Register	24DSI12 specific
DSI_GSC_NREFCR	Nref Control Register (PLL specific)	24DSI12 specific
DSI_GSC_NVCOCR	Nvco Control Register (PLL specific)	24DSI32 specific
DSI_GSC_PRFR	PLL Reference Frequency Register	
DSI_GSC_RAR	Rate Assignments Register	
DSI_GSC_RCAR	Rate Control A Register	24DSI12 specific
DSI_GSC_RCBR	Rate Control B Register	24DSI12 specific
DSI_GSC_RCR	Rate Control Register (LEGACY specific)	24DSI32 specific
DSI_GSC_RDR	Rate Divisors Register	
DSI_GSC_RFCR	Range and Filter Control Register	24DSI32 specific

6.1.2.2. PCI Configuration Registers

Access to the PCI registers is seldom required so these registers are not listed here. For the complete list of the PCI register identifiers refer to the driver header file `gsc_pci9080.h`, which is automatically included via `24dsi.h`.

6.1.2.3. PLX PCI9080 Feature Set Registers

Access to the PLX registers is seldom required so these registers are not listed here. For the complete list of the PLX register identifiers refer to the driver header file `gsc_pci9080.h`, which is automatically included via `24dsi.h`.

6.2. Functions

The driver interface includes the following functions.

6.2.1. close()

This function is the entry point to close a connection to an open 24DSI board.

Prototype

```
int close(int fd);
```

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

Return Value	Description
-1	An error occurred. Consult <code>errno</code> .
0	The operation succeeded.

Example

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

#include "24dsi_dsl.h"

int dsi_dsl_close(int fd)
{
    int err;
    int status;

    status = close(fd);

    if (status == -1)
        printf("ERROR: close() failure, errno = %d\n", errno);

    err = (status == -1) ? 1 : 0;
    return(err);
}
```

6.2.2. ioctl()

This function is the entry point to performing setup and control operations on a 24DSI board. This function should only be called after a successful open of the respective device. The specific operation performed varies according to the `request` argument. The `request` argument also governs the use and interpretation of any additional arguments. The set of supported IOCTL services is defined in section 6.3 beginning on page 31.

NOTE: Many 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 ioctl(int fd, int request, ...);
```

Argument	Description
fd	This is the file descriptor of the device to access.
request	This specifies the desired operation to be performed.
...	This is any additional arguments. If request does not call for any additional arguments, then any additional arguments provided are ignored. The 24DSI IOCTL services use at most one argument.

Return Value	Description
-1	An error occurred. Consult errno.
0	The operation succeeded.

Example

```
#include <errno.h>
#include <stdio.h>
#include <sys/ioctl.h>

#include "24dsi_dsl.h"

int dsi_dsl_ioctl(int fd, int request, void *arg)
{
    int err;
    int status;

    status = ioctl(fd, request, arg);

    if (status == -1)
        printf("ERROR: ioctl() failure, errno = %d\n", errno);

    err = (status == -1) ? 1 : 0;
    return(err);
}
```

6.2.3. open()

This function is the entry point to open a connection to a 24DSI board. The pathname to a 24DSI device node is /dev/24dsin, where the trailing "n" is the zero based index of the board to access.

Prototype

```
int open(const char* pathname, int flags);
```

Argument	Description
pathname	This is the name of the device to open.
flags	This is the desired read/write access. Use O_RDWR.

NOTE: Another form of the `open()` function has a mode argument. This form is not displayed here as the mode argument is ignored when opening an existing file/device.

Return Value	Description
-1	An error occurred. Consult <code>errno</code> .
else	A valid file descriptor.

Example

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

#include "24dsi_dsl.h"

int dsi_dsl_open(unsigned int board)
{
    int    fd;
    char   name[80];

    sprintf(name, DSI_DEV_BASE_NAME "%u", board);
    fd = open(name, O_RDWR);

    if (fd == -1)
    {
        printf("ERROR: open() failure on %s, errno = %d\n",
               name,
               errno);
    }

    return(fd);
}
```

6.2.4. read()

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

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 0 on page 33.

Prototype

```
int read(int fd, void *buf, size_t count);
```

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

Return Value	Description
-1	An error occurred. Consult <code>errno</code> .
0 to count	The operation succeeded. For blocking I/O a return value less than <code>count</code> indicates that the request timed out. For non-blocking I/O a return value less than <code>count</code> indicates that the operation ended prematurely.

Example

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

#include "24dsi_dsl.h"

int dsi_dsl_read(int fd, __u32 *buf, size_t samples)
{
    size_t bytes;
    int status;

    bytes = samples * 4;
    status = read(fd, buf, bytes);

    if (status == -1)
        printf("ERROR: read() failure, errno = %d\n", errno);
    else
        status /= 4;

    return(status);
}
```

6.3. IOCTL Services

The 24DSI driver implements the following IOCTL services. Each service is described along with the applicable `ioctl()` function arguments. In the definitions given the optional argument is identified as `arg`. Unless otherwise stated the return value definitions are those defined for the `ioctl()` function call and any error codes are accessed via `errno`.

6.3.1. DSI_IOCTL_ADC_RATE_OUT

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

Usage

ioctl() 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.

6.3.2. DSI_IOCTL_AIN_BUF_CLEAR

This service immediately clears the current content from the input buffer. It also clears the board's overrun and under run status. This service does not halt sampling.

Usage

ioctl() 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 7.3 on page 57.

6.3.3. DSI_IOCTL_AIN_BUF_INPUT

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

Usage

ioctl() 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 7.3 on page 57.

6.3.4. DSI_IOCTL_AIN_BUF_OVERFLOW

This service operates on the Analog Input Overflow status.

Usage

ioctl() 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.

6.3.5. 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

ioctl() 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.

6.3.6. DSI_IOCTL_AIN_BUF_UNDERFLOW

This service operates on the Analog Input Underflow status.

Usage

ioctl() 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.

6.3.7. DSI_IOCTL_AIN_COUPLING

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

Usage

ioctl() 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.

6.3.8. 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 6.3.10 on page 35.

Usage

ioctl() 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.

6.3.9. 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 6.3.10 on page 35.

Usage

ioctl() 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.

6.3.10. 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 0 on page 33.

Usage

ioctl() 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.

6.3.11. DSI_IOCTL_AIN_MODE

This service selects the analog input signal configuration mode.

Usage

ioctl() 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.

6.3.12. DSI_IOCTL_AIN_RANGE

This service sets the analog input voltage range.

Usage

ioctl() 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.

6.3.13. 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 6.3.14 on page 37.

Usage

ioctl() 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.

6.3.14. 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 6.3.12 on page 35.

Usage

ioctl() 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.

6.3.15. DSI_IOCTL_AUTO_CALIBRATE

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

NOTE: Do not access the board while an auto-calibration cycle is in progress. Doing so may produce indeterminate results, and may lockup the board.

Usage

ioctl() Argument	Description
request	DSI_IOCTL_AUTO_CALIBRATE
arg	Not used.

6.3.16. 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

ioctl() 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.

6.3.17. 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

ioctl() 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.

6.3.18. DSI_IOCTL_DATA_FORMAT

This service sets the data encoding format.

Usage

ioctl() 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.

6.3.19. DSI_IOCTL_DATA_WIDTH

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

Usage

ioctl() 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.

6.3.20. DSI_IOCTL_EXT_CLK_SRC

This service configures the source for the external clock output.

Usage

ioctl() 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.

6.3.21. DSI_IOCTL_EXT_TRIG

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

Usage

ioctl() 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.

6.3.22. DSI_IOCTL_GPS_ENABLE

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

Usage

ioctl() 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.

6.3.23. DSI_IOCTL_GPS_LOCKED

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

Usage

ioctl() 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.

6.3.24. 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

ioctl() 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.

6.3.25. 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

ioctl() 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.

6.3.26. 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

ioctl() 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.

6.3.27. DSI_IOCTL_GPS_TOLERANCE

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

Usage

ioctl() 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.

6.3.28. DSI_IOCTL_INIT_MODE

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

Usage

ioctl() 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.

6.3.29. 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.

Usage

ioctl() Argument	Description
request	DSI_IOCTL_INITIALIZE
arg	Not used.

6.3.30. DSI_IOCTL_IRQ_SEL

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

Usage

ioctl() 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_AUTO_CAL_DONE	This refers to Auto-Calibration completion.
DSI_IRQ_CHAN_READY	This refers to assertion of the Channel Ready status.
DSI_IRQ_INIT_DONE	This refers to initialization completion.

6.3.31. 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

ioctl() Argument	Description
request	DSI_IOCTL_PLL_REF_FREQ_HZ
arg	__s32*

6.3.32. DSI_IOCTL_QUERY

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

Usage

ioctl() Argument	Description
request	DSI_IOCTL_QUERY
arg	__s32*

Valid argument values are as follows.

Value	Description
DSI_QUERY_AUTO_CAL_MS	This returns the maximum duration of the Auto Calibration cycle in milliseconds. If this is zero, then the board doesn't support auto-calibration.
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_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_RFCR	Does the board support the Range and Filter Control Register? (0 = no, 1 = yes)

6.3.33. 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

ioctl() 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 6.3.32 on page 42. Using the value -1 will return the current setting.

6.3.34. 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

ioctl() 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 6.3.32 on page 42. Using the value -1 will return the current setting.

6.3.35. 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

ioctl() 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 6.3.32 on page 42. Using the value -1 will return the current setting.

6.3.36. 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

ioctl() 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 6.3.32 on page 42. Using the value -1 will return the current setting.

6.3.37. 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

ioctl() Argument	Description
request	DSI_IOCTL_REG_MOD
arg	gsc_reg_t*

Definition

```
typedef struct
{
    __u32    reg;
    __u32    value;
    __u32    mask;
} gsc_reg_t;
```

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

6.3.38. 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

ioctl() Argument	Description
request	DSI_IOCTL_REG_READ
arg	gsc_reg_t*

Definition

```
typedef struct
{
    __u32    reg;
    __u32    value;
    __u32    mask;
} gsc_reg_t;
```

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

6.3.39. 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

ioctl() Argument	Description
request	DSI_IOCTL_REG_WRITE
arg	gsc_reg_t*

Definition

```
typedef struct
{
    __u32    reg;
    __u32    value;
    __u32    mask;
} gsc_reg_t;
```

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

6.3.40. DSI_IOCTL_RX_IO_ABORT

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

Usage

ioctl() Argument	Description
request	DSI_IOCTL_RX_IO_ABORT
arg	__s32*

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.

6.3.41. 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 0 on page 33.

Usage

<code>ioctl()</code> Argument	Description
<code>request</code>	<code>DSI_IOCTL_RX_IO_MODE</code>
<code>arg</code>	<code>__s32*</code>

Valid argument values are as follows.

Value	Description
-1	Retrieve the current setting.
<code>GSC_IO_MODE_DMA</code>	Use non-Demand Mode DMA.
<code>GSC_IO_MODE_DMDMA</code>	Use Demand Mode DMA (transfer data as it becomes possible to do so).
<code>GSC_IO_MODE_PIO</code>	Use PIO mode, which is repetitive register access.

6.3.42. 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

<code>ioctl()</code> Argument	Description
<code>request</code>	<code>DSI_IOCTL_RX_IO_OVERFLOW</code>
<code>arg</code>	<code>__s32*</code>

Valid argument values are as follows.

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

6.3.43. DSI_IOCTL_RX_IO_TIMEOUT

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

Usage

<code>ioctl()</code> Argument	Description
<code>request</code>	<code>DSI_IOCTL_RX_IO_TIMEOUT</code>

arg	__s32*
-----	--------

Valid argument values are in the range from zero to 3600, and -1. A value of zero tells the driver not to sleep in order to wait for more sample data, and should only be used with PIO mode reads. A value of -1 is used to retrieve the current setting.

6.3.44. 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

ioctl() 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.

6.3.45. 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 6.3.46 on page 49). When initiating this operation it is the application's responsibility to wait for the Channel Ready bit to be asserted.

Usage

ioctl() Argument	Description
request	DSI_IOCTL_SW_SYNC
arg	Not used.

6.3.46. DSI_IOCTL_SW_SYNC_MODE

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

Usage

ioctl() 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 7.3 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.
--------------------------	---

6.3.47. 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

ioctl() 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.

6.3.48. DSI_IOCTL_WAIT_CANCEL

This service resumes all threads blocked via DSI_IOCTL_WAIT_EVENT IOCTL calls (section 6.3.49, page 51), 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

ioctl() 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 6.3.49.2 on page 52.
gsc	This specifies the set of DSI_WAIT_GSC_* events whose wait requests are to be

	cancelled. Refer to section 6.3.49.3 on page 52.
alt	This is unused by the 24DSI driver and should be zero.
io	This specifies the set of GSC_WAIT_IO_* events whose wait requests are to be cancelled. Refer to section 6.3.49.4 on page 52.
timeout_ms	This is unused by wait cancel operations.
count	Upon return this indicates the number of waits that were cancelled.

6.3.49. 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: 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

ioctl() 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 6.3.49.1 on page 52.
main	This specifies any number of GSC_WAIT_MAIN_* events that the thread is to wait for. Refer to section 6.3.49.2 on page 52.
gsc	This specifies any number of DSI_WAIT_GSC_* events that the thread is to wait for. Refer to section 6.3.49.3 on page 52.
alt	This is unused by the 24DSI driver and must be zero.
io	This specifies any number of GSC_WAIT_IO_* events that the thread is to wait for. Refer to section 6.3.49.4 on page 52.
timeout_ms	This specified the maximum amount of time, in milliseconds, that the thread is to wait for any of the referenced events. This must be greater than zero. Upon return the value will be the approximate amount of time actually waited.
count	This is unused by wait event operations and must be zero.

6.3.49.1. `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 option 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.

6.3.49.2. `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.

6.3.49.3. `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 6.3.30, page 42).

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_AUTO_CAL_DONE</code>	This refers to Auto-Calibration 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.

6.3.49.4. `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
<code>GSC_WAIT_IO_RX_ABORT</code>	This refers to read requests which have been aborted.
<code>GSC_WAIT_IO_RX_DONE</code>	This refers to read requests which have been satisfied.
<code>GSC_WAIT_IO_RX_ERROR</code>	This refers to read requests which end due to an error.
<code>GSC_WAIT_IO_RX_TIMEOUT</code>	This refers to read requests which end due to the timeout period lapse.

6.3.50. DSI_IOCTL_WAIT_STATUS

This service count all threads blocked via the DSI_IOCTL_WAIT_EVENT IOCTL service (section 6.3.49, page 51), 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

ioctl() 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 GSC_WAIT_MAIN_* events whose wait requests are to be counted. Refer to section 6.3.49.2 on page 52.
gsc	This specifies the set of DSI_WAIT_GSC_* events whose wait requests are to be counted. Refer to section 6.3.49.3 on page 52.
alt	This is unused by the 24DSI driver and should be zero.
io	This specifies the set of GSC_WAIT_IO_* events whose wait requests are to be counted. Refer to section 6.3.49.4 on page 52.
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.

6.3.51. DSI_IOCTL_XCVR_TYPE

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

Usage

ioctl() 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.

7. Operation

This section explains some operational procedures using the board. This is in no way intended to be a comprehensive guide on using the 24DSI. This is simply to address a very few issues relating to the board's use.

7.1. Data Reception

Data reception is essentially a three-step process. A simplified version of this process is outlined below.

NOTE: These steps are guidelines only. The actual steps needed may vary significantly from one application to another.

1. Initialize the board to put the 24DSI in a known state.
2. Perform the steps required for any desired input voltage range, number of channels, scan rate settings, etc.
3. Read the data as it is recorded according to the application's needs.

7.1.1.1. No DMA

This is called Programmed I/O or PIO. The driver will read data from the data register until either the buffer is full, or the I/O timeout expires, whichever occurs first.

7.1.1.2. Regular DMA

For a regular DMA transaction, the driver needs to determine how much data to transfer. The driver is set up to only do a DMA operation when the input buffer contains at least `BUFFER_THRESHOLD` samples in the buffer. So if the flags indicate that there is greater than `BUFFER_THRESHOLD` samples available, the driver immediately initiates a DMA transfer between the hardware and the application's buffer. The driver prepares for the interrupt then sleeps until the DMA Done interrupt is received. At that point the service returns. If the flags indicate that there is not enough data in the buffer, the driver sets up and waits for a `BUFFER_THRESHOLD` interrupt. When the interrupt is received, the driver then sets up a DMA transfer as described above. Using this DMA mode, the DMA is initiated only after the data has been received.

7.1.1.3. Demand Mode DMA

This DMA mode is similar to the regular mode, except that the transfer is initiated immediately even if the threshold flag is not yet set. Here however, the actual movement of data occurs as the data becomes available in the data buffer. When the DMA Done interrupt occurs the service returns.

7.2. 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 Two Board Sync sample application of section 5.8 on page 23.

7.2.1. 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 1. If there are only two boards in the setup, then a Clock Driver board is not needed, as illustrated in Figure 2. The table below shows the board programming that is specific to the *star* configuration. See the Two Board Sync sample application source code for additional programming requirements (section 5.8, page 23).

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

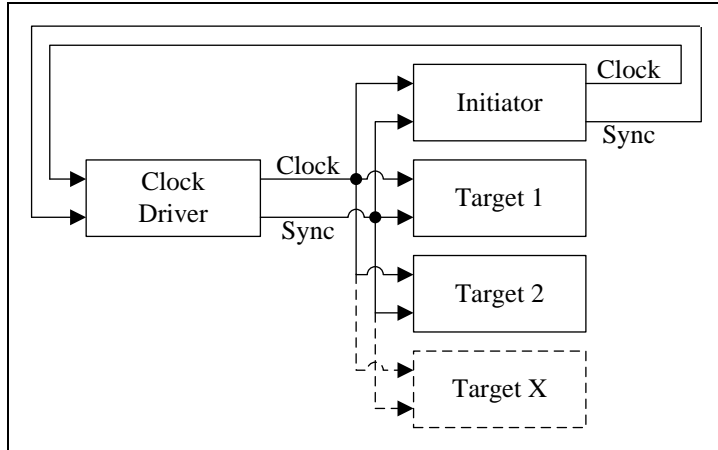


Figure 1 The *star* configuration with three or more boards requires a Clock Driver board.

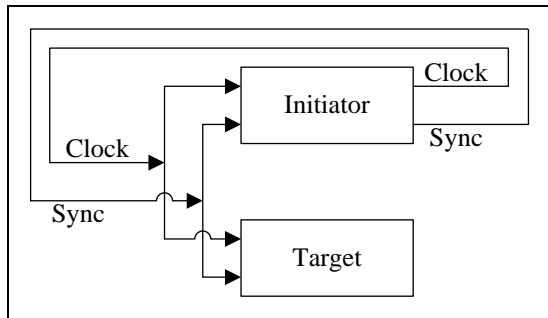


Figure 2 The *star* configuration with only two boards does not require a Clock Driver board.

7.2.2. 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 3. The table below shows the board programming that is specific to the *daisy chain* configuration. See the Two Board Sync sample application source code for additional programming requirements (section 5.8, page 23).

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)

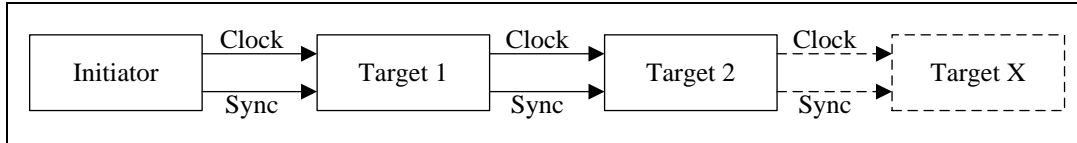


Figure 3 In this configuration the clock and sync signals are daisy chained from one board to the next.

7.3. Clearing the Input Buffer

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

7.3.1. Clear Immediately

One method of clearing the buffer is to use the Clear Buffer IOCTL service (see `DSI_IOCTL_AIN_BUF_CLEAR` in section 6.3.1 on page 31). 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.

7.3.2. 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 0 on page 41). Second, configure the board to clear the buffer when there is a Software Sync pulse (see `DSI_IOCTL_SW_SYNC_MODE` in section 6.3.46 on page 49). Finally, to clear the buffer, initiate a Software Sync pulse (see `DSI_IOCTL_SW_SYNC` in section 6.3.45 on page 49). 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 7.1.1.1 on page 55.)

Document History

Revision	Description
December 22, 2011	Updated to release version 3.9.34.0.
November 14, 2011	Updated to release version 3.8.32.0.
October 12, 2011	Updated to release 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 Auto Calibration related software items. Added the signals sample application.
June 4, 2010	Updated to release version 3.7.16.0. Added the IRQ_SEL and WAIT IOCTL services.
December 28, 2009	Updated to release version 3.6.13.0.
June 23, 2009	Updated to release version 3.5.8.0.
May 28, 2009	Updated to release version 3.4.7.0.
April 22, 2009	Updated to release version 3.3.5.0.
April 17, 2009	Updated to release version 3.2.5.0.
April 16, 2009	Updated to release version 3.2.4.0.
February 27, 2009	Updated to release version 3.1.2.0. Added support for the 24DSI6LN boards.
December 11, 2008	Initial driver release.