

# **18AI32SSC1M**

**18/16-bit, 32 channel, 1M S/S/Ch A/D Input**

**PMC66-18AI32SSC1M**

## **Linux Device Driver User Manual**

**Manual Revision: December 20, 2011  
Driver Release Version 1.5.34.0**

**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](mailto:sales@generalstandards.com)**

**E-mail: [support@generalstandards.com](mailto:support@generalstandards.com)**

## Preface

Copyright © 2009-2011, **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](mailto: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

<b>1. Introduction.....</b>	<b>6</b>
1.1. Purpose.....	6
1.2. Acronyms.....	6
1.3. Definitions .....	6
1.4. Software Overview .....	6
1.5. Hardware Overview .....	6
1.6. Reference Material.....	7
<b>2. Installation.....</b>	<b>8</b>
2.1. CPU and Kernel Support.....	8
2.1.1. 32-bit Support Under 64-bit Environments .....	8
2.2. The /proc File System .....	8
2.3. File List .....	9
2.4. Directory Structure.....	9
2.5. Installation .....	9
2.6. Removal.....	10
2.7. Overall Make Script.....	10
<b>3. The Driver.....</b>	<b>11</b>
3.1. Build .....	11
3.2. Startup.....	11
3.2.1. Manual Driver Startup Procedures .....	11
3.2.2. Automatic Driver Startup Procedures.....	12
3.3. Verification .....	12
3.4. Version.....	12
3.5. Shutdown .....	13
<b>4. Document Source Code Examples.....</b>	<b>14</b>
4.1. Build .....	14
4.2. Library Use .....	14
<b>5. Sample Applications .....</b>	<b>15</b>
5.1. id - Identify Board.....	15
5.1.1. Build .....	15
5.1.2. Execute .....	15
5.2. regs - Register Access .....	16
5.2.1. Build .....	16
5.2.2. Execute .....	16
5.3. sbtest - Single Board Test .....	17
5.3.1. Build .....	17

5.3.2. Execute .....	17
5.4. rx_rate - Receive Rate.....	18
5.4.1. Build .....	18
5.4.2. Execute .....	18
5.5. savedata - Save Acquired Data .....	19
5.5.1. Build .....	19
5.5.2. Execute .....	19
<b>6. Driver Interface.....</b>	<b>20</b>
6.1. Macros .....	20
6.1.1. IOCTL .....	20
6.1.2. Registers .....	20
6.2. Functions.....	21
6.2.1. close() .....	21
6.2.2. ioctl() .....	22
6.2.3. open().....	22
6.2.4. read() .....	23
6.3. IOCTL Services .....	24
6.3.1. AI32SSC1M_IOCTL_AIN_BUF_CLEAR.....	24
6.3.2. AI32SSC1M_IOCTL_AIN_BUF_LEVEL .....	25
6.3.3. AI32SSC1M_IOCTL_AIN_BUF_OVERFLOW.....	25
6.3.4. AI32SSC1M_IOCTL_AIN_BUF_THR_LVL .....	25
6.3.5. AI32SSC1M_IOCTL_AIN_BUF_THR_STS .....	25
6.3.6. AI32SSC1M_IOCTL_AIN_BUF_UNDERFLOW .....	26
6.3.7. AI32SSC1M_IOCTL_AIN_MODE.....	26
6.3.8. AI32SSC1M_IOCTL_AIN_RANGE.....	27
6.3.9. AI32SSC1M_IOCTL_AUTO_CAL_BG.....	27
6.3.10. AI32SSC1M_IOCTL_AUTO_CALIBRATE .....	27
6.3.11. AI32SSC1M_IOCTL_AUX_CLK_MODE .....	28
6.3.12. AI32SSC1M_IOCTL_AUX_IN_POL .....	28
6.3.13. AI32SSC1M_IOCTL_AUX_NOISE .....	28
6.3.14. AI32SSC1M_IOCTL_AUX_OUT_POL .....	29
6.3.15. AI32SSC1M_IOCTL_AUX_SYNC_MODE.....	29
6.3.16. AI32SSC1M_IOCTL_BURST_BUSY .....	29
6.3.17. AI32SSC1M_IOCTL_BURST_SIZE .....	30
6.3.18. AI32SSC1M_IOCTL_BURST_SYNC .....	30
6.3.19. AI32SSC1M_IOCTL_CHAN_ACTIVE.....	30
6.3.20. AI32SSC1M_IOCTL_CHAN_FIRST .....	31
6.3.21. AI32SSC1M_IOCTL_CHAN_LAST .....	31
6.3.22. AI32SSC1M_IOCTL_CHAN_SINGLE .....	31
6.3.23. AI32SSC1M_IOCTL_CLOCK_ENABLE .....	31
6.3.24. AI32SSC1M_IOCTL_DATA_FORMAT .....	32
6.3.25. AI32SSC1M_IOCTL_DATA_PACKING .....	32
6.3.26. AI32SSC1M_IOCTL_DATA_WIDTH .....	33
6.3.27. AI32SSC1M_IOCTL_INITIALIZE.....	33
6.3.28. AI32SSC1M_IOCTL_INPUT_SYNC .....	33
6.3.29. AI32SSC1M_IOCTL_IO_INV .....	33
6.3.30. AI32SSC1M_IOCTL_IRQ0_SEL.....	34
6.3.31. AI32SSC1M_IOCTL_IRQ1_SEL.....	34
6.3.32. AI32SSC1M_IOCTL_PRETRIG.....	34
6.3.33. AI32SSC1M_IOCTL_PRETRIG_DELAY.....	35
6.3.34. AI32SSC1M_IOCTL_PRETRIG_LATCH.....	35
6.3.35. AI32SSC1M_IOCTL_QUERY .....	35

6.3.36. AI32SSC1M_IOCTL_RAG_ENABLE.....	37
6.3.37. AI32SSC1M_IOCTL_RAG_NRATE.....	37
6.3.38. AI32SSC1M_IOCTL_RBG_CLK_SRC.....	37
6.3.39. AI32SSC1M_IOCTL_RBG_ENABLE.....	37
6.3.40. AI32SSC1M_IOCTL_RBG_NRATE.....	38
6.3.41. AI32SSC1M_IOCTL_REG_MOD.....	38
6.3.42. AI32SSC1M_IOCTL_REG_READ.....	39
6.3.43. AI32SSC1M_IOCTL_REG_WRITE.....	39
6.3.44. AI32SSC1M_IOCTL_RX_IO_ABORT.....	40
6.3.45. AI32SSC1M_IOCTL_RX_IO_MODE.....	40
6.3.46. AI32SSC1M_IOCTL_RX_IO_OVERFLOW.....	40
6.3.47. AI32SSC1M_IOCTL_RX_IO_TIMEOUT.....	41
6.3.48. AI32SSC1M_IOCTL_RX_IO_UNDERFLOW.....	41
6.3.49. AI32SSC1M_IOCTL_SAMP_CLK_SRC.....	41
6.3.50. AI32SSC1M_IOCTL_SCAN_MARKER.....	42
6.3.51. AI32SSC1M_IOCTL_WAIT_CANCEL.....	42
6.3.52. AI32SSC1M_IOCTL_WAIT_EVENT.....	43
6.3.53. AI32SSC1M_IOCTL_WAIT_STATUS.....	45
<b>Document History .....</b>	<b>46</b>

# 1. Introduction

This user manual applies to driver release version 1.5.34.0.

## 1.1. Purpose

The purpose of this document is to describe the interface to the 18AI32SSC1M Linux device driver. This driver software provides the interface between "Application Software" and an 18AI32SSC1M 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
DMA	Direct Memory Access
GSC	General Standards Corporation
PMC	PCI Mezzanine Card
RAG	Rate-A Generator
RBG	Rate-B Generator

## 1.3. Definitions

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

Term	Definition
18AI32SSC1M	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 18AI32SSC1M driver software executes under control of the Linux operating system and runs in Kernel Mode as a Kernel Mode device driver. The 18AI32SSC1M 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 18AI32SSC1M is a high-performance, 18-bit analog input board that incorporates up 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 1M samples per second over each channel. Internal clocking permits sampling rates from 1M samples per second down to less than one sample per second. Onboard storage permits data buffering of up to 256K samples, for all channels collectively, between the cable interface and the PCI bus. When sampling data at 16-bits of resolution, data packing within the onboard buffer permits data buffering of up to 512K samples. This allows the 18AI32SSC1M to sustain continuous throughput from the cable interface independent of the PCI bus interface. The 18AI32SSC1M also permits multiple boards to be synchronized so that all boards sample data in unison. In addition, the board includes auto-calibration capability. For lower sampling rates auto-calibration can be performed in the background, thus permitting continuous auto-calibration.

## 1.6. Reference Material

The following reference material may be of particular benefit in using an 18AI32SSC1M 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 *18AI32SSC1M User Manual* from General Standards Corporation.
- The *PCI9056 PCI Bus Master Interface Chip* data handbook from PLX Technology, Inc.

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

## 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
2.6.33	Red Hat Fedora Core 13	Yes	Yes
2.6.31	Red Hat Fedora Core 12	Yes	Yes
2.6.29	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/18ai32ssc1m` file will be "no".

### 2.2. The /proc File System

While the driver is installed, the text file `/proc/18ai32ssc1m` can be read to obtain information about the driver. Each file entry includes an entry name followed immediately by a colon, a space character, and the entry value. Below is an example of what appears in the file, followed by descriptions of each entry.

```
version: 1.5.34
built: Dec 20 2011, 10:46:09
32-bit support: yes (native)
boards: 1
models: 18AI32SSC1M
```



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
18ai32ssc1m.linux.tar.gz	This archive contains the driver, the samples and all related sources.
18ai32ssc1m_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
18ai32ssc1m	This is the source root directory. The user manual and overall make script are placed here.
18ai32ssc1m\docsrc	This directory contains the Document Source Code Examples. Refer to section 4 on page 14.
18ai32ssc1m\driver	This directory contains the driver and its sources. Refer to section 6 on page 20.
18ai32ssc1m\id	This directory contains the Identification application. Refer to section 0 on page 15.
18ai32ssc1m\regs	This directory contains the Register Access application. Refer to section 5.1.2 on page 15.
18ai32ssc1m\rxrate	This directory contains the Receive rate application. Refer to section 5.2.2 on page 16.
18ai32ssc1m\savedata	This directory contains the Save Data application. Refer to section 5.5 on page 19.
18ai32ssc1m\sbtest	This directory contains the Single Board Test application. Refer to section 5.2.2 on page 16.
18ai32ssc1m\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 `18ai32ssc1m.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 `18ai32ssc1m` in the current directory, and then copies all of the archive's files into this new directory.

```
tar -xzf 18ai32ssc1m.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 13.
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 18ai32ssc1m.linux.tar.gz 18ai32ssc1m
```

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

```
rm -f /dev/18ai32ssc1m*
```

5. If the automated startup procedure was adopted (see section 3.2.2 on page 12), then edit the system startup script `rc.local` and remove the line that invokes the 18AI32SSC1M'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/18ai32ssc1m`.
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 `18ai32ssc1m.linux.tar.gz`. The driver's files are summarized in the table below.

File	Description
<code>driver/*.c</code>	These are driver source files.
<code>driver/*.h</code>	These are driver header files.
<code>driver/18ai32ssc1m.h</code>	This is the main driver header file. This header should be included by 18AI32SSC1M applications.
<code>driver/start</code>	This is a shell script to install the driver executable and create the device nodes.
<code>driver/Makefile</code>	This is the driver make file.

#### 3.1. Build

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

Follow the below steps to build the driver.

1. Change to the directory where the driver and its sources are installed, which may be `/usr/src/linux/drivers/18ai32ssc1m/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 sources are installed, which may be `/usr/src/linux/drivers/18ai32ssc1m/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 18AI32SSC1M 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 `18ai32ssclm` 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/18ai32ssclm*
```

### 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/18ai32ssclm/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/18ai32ssclm` 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/18ai32ssclm
```

### 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/18ai32ssclm` 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 18ai32ssc1m`
3. Verify that the driver module has been unloaded by issuing the below command. The module name `18ai32ssc1m` should not be in the listed output.

`lsmod`

## 4. Document Source Code Examples

The archive file `18ai32ssc1m.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 18AI32SSC1M 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 18AI32SSC1M root directory.

File	Description
<code>docsrc/*.c</code>	These are the C source files.
<code>docsrc/18ai32ssc1m_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/18ai32ssc1m/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 `18ai32ssc1m_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/18ai32ssc1m/docsrc`. Link time use also has two requirements. First, include the static library `18ai32ssc1m_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/18ai32ssc1m/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 five minutes to complete, depending on the number of channels on the board. 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. rx\_rate - Receive Rate

This sample console application provides a command line driven Linux application that configures the board then reads data at the fastest transfer rate for the given configuration. Command line switches permit selection of all three data transfer modes. The application's sources are summarized in the below table.

File	Description
rx_rate/*.c	These are the application's source files.
rx_rate/main.h	This is the application's header file.
rx_rate/makefile	This is the application make file.
rx_rate/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 (.../rx\_rate).
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 (.../rx\_rate).
2. Start the sample application by issuing the command given below. Once started the application will automatically performs an initialization of the board, then it will read and save the data. A single iteration should take only a few seconds. The command line arguments are described in the table below.

```
./rx_rate <-c> <-C> <-dma> <-dmdma> <-m#> <-n#> <-pio> <-r#> <index>
```

Argument	Description
-c	Repeat the operation until an error is encountered.
-C	Repeat the operation, but continue even if errors are encountered.
-dma	Perform the data transfer using DMA.
-dmdma	Perform the data transfer using Demand Mode DMA.
-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.
-pio	Perform the data transfer using PIO.
-r#	Retrieve “#” number of megabytes of data, where “#” is a decimal number.
index	This is the zero based index of the board to access.

## 5.5. savedata - Save Acquired Data

This sample console application provides a command line driven Linux application that reads 1MB of data then saves the data in ASCII hex format to a file (data.txt). The application's sources are summarized in the below table.

File	Description
savedata/*.c	These are the application's source files.
savedata/main.h	This is the application's header file.
savedata/makefile	This is the application make file.
savedata/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 (.../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
```

### 5.5.2. 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 will automatically performs an initialization of the board, then it will read and save the data. A single iteration with the default options should take less than 10 seconds to complete. The command line arguments are described in the table below.

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

Argument	Description
-c	Repeat the operation until an error is encountered.
-C	Repeat the operation, but continue even if errors are encountered.
-dma	Perform the data transfer using DMA. This is the default.
-dmdma	Perform the data transfer using Demand Mode DMA.
-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.
-pio	Perform the data transfer using PIO.
index	This is the zero based index of the board to access.

## 6. Driver Interface

The 18AI32SSC1M 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 18AI32SSC1M boards for Linux applications. The interface includes various macros, data types and functions, all of which are described in the following paragraphs. The 18AI32SSC1M specific portion of the driver interface is defined in the header file `18ai32ssc1m.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 `18ai32ssc1m.h`.

#### 6.1.1. IOCTL

The IOCTL macros are documented in 6.3 beginning on page 24.

#### 6.1.2. Registers

The following gives the complete set of 18AI32SSC1M registers.

##### 6.1.2.1. GSC Registers

The following tables give the complete set of GSC specific 18AI32SSC1M registers. For detailed definitions of these registers refer to the relevant 18AI32SSC1M User Manual. Please note that the set of registers supported by any given board may vary according to model and firmware version. For the set of supported registers and detailed definitions of these registers please refer to the appropriate *18AI32SSC1M User Manual*.

Macro	Description
AI32SSC1M_GSC_ACAR	Active Channel Assignment Register
AI32SSC1M_GSC_ASIOCR	Auxiliary Sync I/O Control Register
AI32SSC1M_GSC_AVR	Autocal Values Register
AI32SSC1M_GSC_ARWR	Auxiliary Read/Write Register
AI32SSC1M_GSC_BCFGR	Board Configuration Register
AI32SSC1M_GSC_BCTLR	Board Control Register
AI32SSC1M_GSC_BSTSR	Burst Size Register
AI32SSC1M_GSC_BUFSR	Buffer Size Register
AI32SSC1M_GSC_DITR	Disable Initial Trigger Register
AI32SSC1M_GSC_IBCR	Input Buffer Control Register
AI32SSC1M_GSC_IBDR	Input Buffer Data Register
AI32SSC1M_GSC_ICR	Interrupt Control Register
AI32SSC1M_GSC_PTCHR	Pretrigger Counter High Register
AI32SSC1M_GSC_PTCLR	Pretrigger Counter Low Register
AI32SSC1M_GSC_RAGR	Rate-A Generator Register
AI32SSC1M_GSC_RBGR	Rate-B Generator Register
AI32SSC1M_GSC_TUR	Test Utility Register
AI32SSC1M_GSC_SMLWR	Scan Marker Lower Word Register
AI32SSC1M_GSC_SMUWR	Scan Marker Upper Word Register
AI32SSC1M_GSC_SSCR	Scan & Sync Control Register

### 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_pci9056.h`, which is automatically included via `18ai32ssclm.h`.

### 6.1.2.3. PLX PCI9056 Feature Set Registers

Access to the PLX registers is seldom required so these registers are not listed here. For the complete list of the PLX register identifiers refer to the driver header file `gsc_pci9056.h`, which is automatically included via `18ai32ssclm.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 18AI32SSC1M board.

Prototype

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

Argument	Description
<code>fd</code>	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 "18ai32ssclm_dsl.h"

int ai32ssclm_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 an 18AI32SSC1M 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 24.

#### Prototype

```
int ioctl(int fd, int request, ...);
```

Argument	Description
<code>fd</code>	This is the file descriptor of the device to access.
<code>request</code>	This specifies the desired operation to be performed.
<code>...</code>	This is any additional arguments. If <code>request</code> does not call for any additional arguments, then any additional arguments provided are ignored. The 18AI32SSC1M IOCTL services use at most one argument.

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

#### Example

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

#include "18ai32ssclm_dsl.h"

int ai32ssclm_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 an 18AI32SSC1M board. The pathname to an 18AI32SSC1M device node is `/dev/18ai32ssclmn`, 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 "18ai32ssclm_dsl.h"

int ai32ssclm_dsl_open(unsigned int board)
{
    int    fd;
    char   name[80];

    sprintf(name, AI32SSC1M_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 18AI32SSC1M. 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.

#### 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 count indicates that the request timed out. For non-blocking I/O a return value less than count indicates that the operation ended prematurely.
------------	--

**Example**

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

#include "18ai32ssc1m_dsl.h"

int ai32ssc1m_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 18AI32SSC1M 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. AI32SSC1M\_IOCTL\_AIN\_BUF\_CLEAR**

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

**Usage**

<b>ioctl() Argument</b>	<b>Description</b>
<code>request</code>	<code>AI32SSC1M_IOCTL_AIN_BUF_CLEAR</code>
<code>arg</code>	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. To clear the input buffer on a scan boundary ADC processing must be disabled first (see `AI32SSC1M_IOCTL_CLOCK_ENABLE`, section 6.3.23, page 31).



### 6.3.2. AI32SSC1M\_IOCTL\_AIN\_BUF\_LEVEL

This service returns the current number of 32-bit data items in the input buffer. This may equal the number of A/D data values in the buffer, depending on the board's configuration.

#### Usage

<b>ioctl( ) Argument</b>	<b>Description</b>
request	AI32SSC1M_IOCTL_AIN_BUF_LEVEL
arg	__s32*

The value returned will be from zero to 256K (262,144).

### 6.3.3. AI32SSC1M\_IOCTL\_AIN\_BUF\_OVERFLOW

This service operates on the Input Buffer Overflow status.

#### Usage

<b>ioctl( ) Argument</b>	<b>Description</b>
request	AI32SSC1M_IOCTL_AIN_BUF_OVERFLOW
arg	__s32*

Valid argument values supplied to the service are as follows.

<b>Value</b>	<b>Description</b>
-1	Retrieve the current state.
AI32SSC1M_AIN_BUF_OVERFLOW_CLEAR	Clear the overflow status.
AI32SSC1M_AIN_BUF_OVERFLOW_IGNORE	Ignore the current status.

The current state is reported as one of the following values.

<b>Value</b>	<b>Description</b>
AI32SSC1M_AIN_BUF_OVERFLOW_NO	The buffer has experienced an overflow condition.
AI32SSC1M_AIN_BUF_OVERFLOW_YES	The buffer has not experienced an overflow condition.

### 6.3.4. AI32SSC1M\_IOCTL\_AIN\_BUF\_THR\_LVL

This service configures the input buffer threshold level.

#### Usage

<b>ioctl( ) Argument</b>	<b>Description</b>
request	AI32SSC1M_IOCTL_AIN_BUF_THR_LVL
arg	__s32*

Valid argument values are from zero to 0x3FFFF, and -1. A value of -1 will return the current threshold level setting.

### 6.3.5. AI32SSC1M\_IOCTL\_AIN\_BUF\_THR\_STS

This service retrieves the current input buffer threshold level status, which indicates whether or not there are more than Threshold Level number of 32-bit data items in the input buffer.

## Usage

<b>ioctl() Argument</b>	<b>Description</b>
request	AI32SSC1M_IOCTL_AIN_BUF_THR_STS
arg	__s32*

The current status is reported as one of the following values.

<b>Value</b>	<b>Description</b>
AI32SSC1M_AIN_BUF_THR_STS_CLEAR	The buffer contains Threshold Level number of data items, or fewer.
AI32SSC1M_AIN_BUF_THR_STS_SET	The buffer contains more than Threshold Level number of data items.

### 6.3.6. AI32SSC1M\_IOCTL\_AIN\_BUF\_UNDERFLOW

This service operates on the Input Buffer Underflow status.

## Usage

<b>ioctl() Argument</b>	<b>Description</b>
request	AI32SSC1M_IOCTL_AIN_BUF_UNDERFLOW
arg	__s32*

Valid argument values supplied to the service are as follows.

<b>Value</b>	<b>Description</b>
-1	Retrieve the current state.
AI32SSC1M_AIN_BUF_UNDERFLOW_CLEAR	Clear the underflow status.
AI32SSC1M_AIN_BUF_UNDERFLOW_IGNORE	Ignore the current status.

Valid argument values are as follows.

<b>Value</b>	<b>Description</b>
AI32SSC1M_AIN_BUF_UNDERFLOW_NO	The buffer has experienced an underflow condition.
AI32SSC1M_AIN_BUF_UNDERFLOW_YES	The buffer has not experienced an underflow condition.

### 6.3.7. AI32SSC1M\_IOCTL\_AIN\_MODE

This service configures the board's Analog Input Mode.

## Usage

<b>ioctl() Argument</b>	<b>Description</b>
request	AI32SSC1M_IOCTL_AIN_MODE
arg	__s32*

Valid argument values are as follows.

<b>Value</b>	<b>Description</b>
-1	Retrieve the current setting.
AI32SSC1M_AIN_MODE_DIFF	Configure the input channels for differential operation.
AI32SSC1M_AIN_MODE_VREF	Connect the input channels to the onboard VREF signal.
AI32SSC1M_AIN_MODE_ZERO	Connect the input channels to the onboard zero voltage signal.

**6.3.8. AI32SSC1M\_IOCTL\_AIN\_RANGE**

This service configures the analog input voltage range.

**Usage**

<b>ioctl() Argument</b>	<b>Description</b>
request	AI32SSC1M_IOCTL_AIN_RANGE
arg	__s32*

Valid argument values are as follows.

<b>Value</b>	<b>Description</b>
-1	Retrieve the current setting.
AI32SSC1M_AIN_RANGE_1_25V	Set the input voltage range to $\pm 1.25$ volts. This option is valid only on lower voltage range boards. *
AI32SSC1M_AIN_RANGE_2_5V	Set the input voltage range to $\pm 2.5$ volts. This option is valid only on lower voltage range boards. *
AI32SSC1M_AIN_RANGE_5V	Set the input voltage range to $\pm 5$ volts. This option is valid only on higher voltage range boards. *
AI32SSC1M_AIN_RANGE_10V	Set the input voltage range to $\pm 10$ volts. This option is valid only on higher voltage range boards. *

\* The voltage range query option (AI32SSC1M\_QUERY\_V\_RANGE) can be used to determine the range supported by the board. Refer to section 6.3.32 on page 34.

**6.3.9. AI32SSC1M\_IOCTL\_AUTO\_CAL\_BG**

This service enables or disables background auto-calibration operation. Most configuration settings should be made before background auto-calibration begins.

**Usage**

<b>ioctl() Argument</b>	<b>Description</b>
request	AI32SSC1M_IOCTL_AUTO_CAL_BG
arg	__s32*

Valid argument values are as follows.

<b>Value</b>	<b>Description</b>
-1	Retrieve the current setting.
AI32SSC1M_AUTO_CAL_BG_DISABLE	This disables background auto-calibration operation.
AI32SSC1M_AUTO_CAL_BG_ENABLE	This enables background auto-calibration operation.

**6.3.10. AI32SSC1M\_IOCTL\_AUTO\_CALIBRATE**

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

**Usage**

<b>ioctl() Argument</b>	<b>Description</b>
request	AI32SSC1M_IOCTL_AUTO_CALIBRATE
arg	Not used.

**6.3.11. AI32SSC1M\_IOCTL\_AUX\_CLK\_MODE**

This service configures the clock signal on the board's auxiliary signal connector.

**Usage**

<b>ioctl( ) Argument</b>	<b>Description</b>
request	AI32SSC1M_IOCTL_AUX_CLK_MODE
arg	__s32*

Valid argument values are as follows.

<b>Value</b>	<b>Description</b>
-1	Retrieve the current setting.
AI32SSC1M_AUX_CLK_MODE_DISABLE	This disables the signal.
AI32SSC1M_AUX_CLK_MODE_INPUT	This configures the signal as an input.
AI32SSC1M_AUX_CLK_MODE_OUTPUT	This configures the signal as an output.

**6.3.12. AI32SSC1M\_IOCTL\_AUX\_IN\_POL**

This service configures the polarity of the input signals on the board's auxiliary signal connector.

**Usage**

<b>ioctl( ) Argument</b>	<b>Description</b>
request	AI32SSC1M_IOCTL_AUX_IN_POL
arg	__s32*

Valid argument values are as follows.

<b>Value</b>	<b>Description</b>
-1	Retrieve the current setting.
AI32SSC1M_AUX_IN_POL_HI_2_LO	Clocking occurs on high-to-low transitions.
AI32SSC1M_AUX_IN_POL_LO_2_HI	Clocking occurs on low-to-high transitions.

**6.3.13. AI32SSC1M\_IOCTL\_AUX\_NOISE**

This service configures the noise sensitivity setting for signals on the board's auxiliary signal connector.

**Usage**

<b>ioctl( ) Argument</b>	<b>Description</b>
request	AI32SSC1M_IOCTL_AUX_NOISE
arg	__s32*

Valid argument values are as follows.

<b>Value</b>	<b>Description</b>
-1	Retrieve the current setting.
AI32SSC1M_AUX_NOISE_HIGH	This refers to high noise sensitivity.
AI32SSC1M_AUX_NOISE_LOW	This refers to low noise sensitivity.

**6.3.14. AI32SSC1M\_IOCTL\_AUX\_OUT\_POL**

This service configures the polarity of the output signals on the board's auxiliary signal connector.

**Usage**

<b>ioctl( ) Argument</b>	<b>Description</b>
request	AI32SSC1M_IOCTL_AUX_OUT_POL
arg	__s32*

Valid argument values are as follows.

<b>Value</b>	<b>Description</b>
-1	Retrieve the current setting.
AI32SSC1M_AUX_OUT_POL_HI_PULSE	The active state is generated via high going pulses.
AI32SSC1M_AUX_OUT_POL_LOW_PULSE	The active state is generated via low going pulses.

**6.3.15. AI32SSC1M\_IOCTL\_AUX\_SYNC\_MODE**

This service configures the sync signal on the board's auxiliary signal connector.

**Usage**

<b>ioctl( ) Argument</b>	<b>Description</b>
request	AI32SSC1M_IOCTL_AUX_SYNC_MODE
arg	__s32*

Valid argument values are as follows.

<b>Value</b>	<b>Description</b>
-1	Retrieve the current setting.
AI32SSC1M_AUX_SYNC_MODE_DISABLE	This disables the signal.
AI32SSC1M_AUX_SYNC_MODE_INPUT	This configures the signal as an input.
AI32SSC1M_AUX_SYNC_MODE_OUTPUT	This configures the signal as an output.

**6.3.16. AI32SSC1M\_IOCTL\_BURST\_BUSY**

This service reports on the board's burst activity state.

**Usage**

<b>ioctl( ) Argument</b>	<b>Description</b>
request	AI32SSC1M_IOCTL_BURST_BUSY
arg	__s32*

The value returned will be one of the following.

<b>Value</b>	<b>Description</b>
AI32SSC1M_BURST_BUSY_ACTIVE	A bursting activity is in progress.
AI32SSC1M_BURST_BUSY_IDLE	No bursting activity is in progress.

**6.3.17. AI32SSC1M\_IOCTL\_BURST\_SIZE**

This service configures the size of a single burst (the count is in scans, which is an A/D conversion of all active channels).

**Usage**

<b>ioctl( ) Argument</b>	<b>Description</b>
request	AI32SSC1M_IOCTL_BURST_SIZE
arg	__s32*

Valid argument values are from zero to 0xFFFFF, or -1 to retrieve the current setting.

**6.3.18. AI32SSC1M\_IOCTL\_BURST\_SYNC**

This service configures the clocking source for burst operations.

**Usage**

<b>ioctl( ) Argument</b>	<b>Description</b>
request	AI32SSC1M_IOCTL_BURST_SYNC
arg	__s32*

Valid argument values are as follows.

<b>Value</b>	<b>Description</b>
-1	Retrieve the current setting.
AI32SSC1M_BURST_SYNC_BCR	Bursting is driven by the Board Control Register's Input Sync bit.
AI32SSC1M_BURST_SYNC_DISABLE	Bursting is disabled.
AI32SSC1M_BURST_SYNC_EXT	Bursting is driven by the cable's Sync Input cable signal.
AI32SSC1M_BURST_SYNC_RBG	Bursting is driven by the Rate-B Generator.

**6.3.19. AI32SSC1M\_IOCTL\_CHAN\_ACTIVE**

This service configures the setting for the number and range of active channels to scan.

**Usage**

<b>ioctl( ) Argument</b>	<b>Description</b>
request	AI32SSC1M_IOCTL_CHAN_ACTIVE
arg	__s32*

Valid argument values are as follows.

<b>Value</b>	<b>Description</b>
-1	Retrieve the current setting.
AI32SSC1M_CHAN_ACTIVE_0_1	This refers to channels zero through one.
AI32SSC1M_CHAN_ACTIVE_0_3	This refers to channels zero through three.
AI32SSC1M_CHAN_ACTIVE_0_7	This refers to channels zero through seven.
AI32SSC1M_CHAN_ACTIVE_0_15	This refers to channels zero through 15.
AI32SSC1M_CHAN_ACTIVE_0_31	This refers to channels zero through 31.
AI32SSC1M_CHAN_ACTIVE_RANGE	This refers to a user specified range of channels from a <i>first</i> selection to a <i>last</i> selection. **

AI32SSC1M_CHAN_ACTIVE_SINGLE	This refers to a single, user specified channel. *
------------------------------	--

- \* The channel selection is specified with the service AI32SSC1M\_IOCTL\_CHAN\_SINGLE (section 6.3.22, page 31).
- \*\* The *first* channel is specified with the service AI32SSC1M\_IOCTL\_CHAN\_FIRST (section 6.3.20, page 31). The *last* channel is specified with the service AI32SSC1M\_IOCTL\_CHAN\_LAST (section 6.3.21, page 31).

### 6.3.20. AI32SSC1M\_IOCTL\_CHAN\_FIRST

This service configures the setting for the first channel to scan when the active channel setting is set to the *range* option (AI32SSC1M\_CHAN\_ACTIVE\_RANGE, section 6.3.19, page 30).

#### Usage

ioctl() Argument	Description
request	AI32SSC1M_IOCTL_CHAN_FIRST
arg	__s32*

Valid argument values are from zero to one less than the current *last* setting, or -1 to retrieve the current setting.

### 6.3.21. AI32SSC1M\_IOCTL\_CHAN\_LAST

This service configures the setting of the last channel to scan when the active channel setting is set to the *range* option (AI32SSC1M\_CHAN\_ACTIVE\_RANGE, section 6.3.19, page 30).

#### Usage

ioctl() Argument	Description
request	AI32SSC1M_IOCTL_CHAN_LAST
arg	__s32*

Valid argument values are from the current *first* setting to one less than the number of channels on the board, or -1 to retrieve the current setting.

### 6.3.22. AI32SSC1M\_IOCTL\_CHAN\_SINGLE

This service configures the setting for the channel to scan when the active channel setting is set to the *single* option (AI32SSC1M\_CHAN\_ACTIVE\_SINGLE, section 6.3.19, page 30).

#### Usage

ioctl() Argument	Description
request	AI32SSC1M_IOCTL_CHAN_SINGLE
arg	__s32*

Valid argument values are from zero to one less than the number of channels on the board, or -1 to retrieve the current setting.

### 6.3.23. AI32SSC1M\_IOCTL\_CLOCK\_ENABLE

This service enables or disables the ADC clocking of data.

## Usage

<b>ioctl( ) Argument</b>	<b>Description</b>
request	AI32SSC1M_IOCTL_CLOCK_ENABLE
arg	__s32*

Valid argument values are as follows.

<b>Value</b>	<b>Description</b>
-1	Retrieve the current setting.
AI32SSC1M_CLOCK_ENABLE_NO	This disables the data clocking process.
AI32SSC1M_CLOCK_ENABLE_YES	This enables the data clocking process.

**6.3.24. AI32SSC1M\_IOCTL\_DATA\_FORMAT**

This service configures the data encoding format.

## Usage

<b>ioctl( ) Argument</b>	<b>Description</b>
request	AI32SSC1M_IOCTL_DATA_FORMAT
arg	__s32*

Valid argument values are as follows.

<b>Value</b>	<b>Description</b>
-1	Retrieve the current setting.
AI32SSC1M_DATA_FORMAT_2S_COMP	This refers to the Twos Compliment data format.
AI32SSC1M_DATA_FORMAT_OFF_BIN	This refers to the Offset Binary encoding format.

**6.3.25. AI32SSC1M\_IOCTL\_DATA\_PACKING**

This service configures the data packing feature.

## Usage

<b>ioctl( ) Argument</b>	<b>Description</b>
request	AI32SSC1M_IOCTL_DATA_PACKING
arg	__s32*

Valid argument values are as follows.

<b>Value</b>	<b>Description</b>
-1	Retrieve the current setting.
AI32SSC1M_DATA_PACKING_DISABLE	This option disables data packing so that A/D values in the input buffer are 32-bits wide.
AI32SSC1M_DATA_PACKING_ENABLE	This option enables data packing so that A/D values in the input buffer are 16-bits wide. *

\* Data packing occurs only with the 16-bit data width setting (see AI32SSC1M\_IOCTL\_DATA\_WIDTH, section 6.3.26, page 33).



**6.3.26. AI32SSC1M\_IOCTL\_DATA\_WIDTH**

This service configures the encoded data width. This service is available only for those boards whose data width is configurable.

**Usage**

<b>ioctl() Argument</b>	<b>Description</b>
request	AI32SSC1M_IOCTL_DATA_WIDTH
arg	__s32*

Valid argument values are as follows.

<b>Value</b>	<b>Description</b>
-1	Retrieve the current setting.
AI32SSC1M_DATA_WIDTH_16	This refers to 16-bit data.
AI32SSC1M_DATA_WIDTH_18	This refers to 18-bit data.

**6.3.27. AI32SSC1M\_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**

<b>ioctl() Argument</b>	<b>Description</b>
request	AI32SSC1M_IOCTL_INITIALIZE
arg	Not used.

**6.3.28. AI32SSC1M\_IOCTL\_INPUT\_SYNC**

This service initiates an Input Sync operation. The driver will wait for completion, but no more than the read timeout period. If the read timeout is zero, then the driver will wait up to one second for completion. (Refer to service AI32SSC1M\_IOCTL\_RX\_IO\_TIMEOUT in section 6.3.47 on page 41.)

**Usage**

<b>ioctl() Argument</b>	<b>Description</b>
request	AI32SSC1M_IOCTL_INPUT_SYNC
arg	Not used.

**6.3.29. AI32SSC1M\_IOCTL\_IO\_INV**

This service configures the inversion of the cable's clock and sync I/O signals.

**Usage**

<b>ioctl() Argument</b>	<b>Description</b>
request	AI32SSC1M_IOCTL_IO_INV
arg	__s32*

Valid argument values are as follows.

Value	Description
-1	Retrieve the current setting.
AI32SSC1M_IO_INV_HIGH	Active signals are asserted high.
AI32SSC1M_IO_INV_LOW	Active signals are asserted low.

### 6.3.30. AI32SSC1M\_IOCTL\_IRQ0\_SEL

This service configures the interrupt source selection for interrupt number zero.

Usage

ioctl( ) Argument	Description
request	AI32SSC1M_IOCTL_IRQ0_SEL
arg	__s32*

Valid argument values are as follows.

Value	Description
-1	Retrieve the current setting.
AI32SSC1M_IRQ0_AUTO_CAL_DONE	This refers to the completion of an auto-calibration cycle.
AI32SSC1M_IRQ0_BURST_DONE	This refers to the completion of an input burst.
AI32SSC1M_IRQ0_BURST_START	This refers to the beginning of an input burst.
AI32SSC1M_IRQ0_INIT_DONE	This refers to the completion of an initialization cycle.
AI32SSC1M_IRQ0_SYNC_DONE	This refers to the completion of a sync operation.
AI32SSC1M_IRQ0_SYNC_START	This refers to the beginning of a sync operation.

### 6.3.31. AI32SSC1M\_IOCTL\_IRQ1\_SEL

This service configures the interrupt source selection for interrupt number one.

Usage

ioctl( ) Argument	Description
request	AI32SSC1M_IOCTL_IRQ1_SEL
arg	__s32*

Valid argument values are as follows.

Value	Description
-1	Retrieve the current setting.
AI32SSC1M_IRQ1_IN_BUF_OVR_UNDR	This refers to the occurrence of either an input buffer overflow or an input buffer underflow.
AI32SSC1M_IRQ1_IN_BUF_THR_H2L	This refers to the input buffer threshold status being negated.
AI32SSC1M_IRQ1_IN_BUF_THR_L2H	This refers to the input buffer threshold status being asserted.
AI32SSC1M_IRQ1_NONE	This disabled the interrupt.

### 6.3.32. AI32SSC1M\_IOCTL\_PRETRIG

This service enables or disabled the Pretrigger feature.

## Usage

<b>ioctl() Argument</b>	<b>Description</b>
request	AI32SSC1M_IOCTL_PRETRIG
arg	__s32*

Valid argument values are as follows.

<b>Value</b>	<b>Description</b>
-1	Retrieve the current setting.
AI32SSC1M_PRETRIG_DISABLE	This disables the Pretrigger feature.
AI32SSC1M_PRETRIG_ENABLE	This enables the Pretrigger feature.

**6.3.33. AI32SSC1M\_IOCTL\_PRETRIG\_DELAY**

This service configures the delay between the clearing of the buffer and the first recognized trigger event.

**NOTE:** This service sets the Cloak bit in the Disable Initial Trigger Register.

## Usage

<b>ioctl() Argument</b>	<b>Description</b>
request	AI32SSC1M_IOCTL_PRETRIG_DELAY
arg	__s32*

Valid argument values are from 1 to 0xFFFF.

**6.3.34. AI32SSC1M\_IOCTL\_PRETRIG\_LATCH**

This service deals with the status of Pretrigger Latch.

## Usage

<b>ioctl() Argument</b>	<b>Description</b>
request	AI32SSC1M_IOCTL_PRETRIG_LATCH
arg	__s32*

Valid argument values are as follows.

<b>Value</b>	<b>Description</b>
-1	Retrieve the current state.
AI32SSC1M_PRETRIG_LATCH_CLEAR	This option clears the latch bit.
AI32SSC1M_PRETRIG_LATCH_IGNORE	This option takes no action regarding the latch bit.

When retrieving the current state the below values are returned.

<b>Value</b>	<b>Description</b>
AI32SSC1M_PRETRIG_LATCH_ACTIVE	The latch bit is set.
AI32SSC1M_PRETRIG_LATCH_IDLE	The latch bit is not set.

**6.3.35. AI32SSC1M\_IOCTL\_QUERY**

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

## Usage

<b>ioctl()</b> Argument	Description
request	AI32SSC1M_IOCTL_QUERY
arg	__s32*

Valid argument values are as follows.

<b>Value</b>	Description
AI32SSC1M_QUERY_AUTO_CAL_BG	This indicates if the board supports the background auto-calibration feature (0 = no, 1 = yes).
AI32SSC1M_QUERY_AUTO_CAL_MS	This returns the maximum duration of the Auto Calibration cycle in milliseconds.
AI32SSC1M_QUERY_CHANNEL_MAX	This returns the maximum number of input channels supported by the board, which may be more than the board's current configuration.
AI32SSC1M_QUERY_CHANNEL_QTY	This returns the actual number of input channels on the current board. If the value returned is -1, then the driver was unable to determine the number of channels.
AI32SSC1M_QUERY_COUNT	This returns the number of query options supported by the IOCTL service.
AI32SSC1M_QUERY_DATA_WIDTH	This indicates if the board supports the feature of configuring the A/D conversion data width (0 = no, 1 = yes).
AI32SSC1M_QUERY_DEVICE_TYPE	This returns the identifier value for the board's type. This should be GSC_DEV_TYPE_18AI32SSC1M.
AI32SSC1M_QUERY_FGEN_MAX	This returns the maximum supported FGEN value.
AI32SSC1M_QUERY_FGEN_MIN	This returns the minimum supported FGEN value.
AI32SSC1M_QUERY_FIFO_SIZE	This returns the size of the input buffer in 32-bit A/D values.
AI32SSC1M_QUERY_FSAMP_MAX	This gives the maximum FSAMP value in S/S.
AI32SSC1M_QUERY_FSAMP_MIN	This gives the minimum FSAMP value in S/S.
AI32SSC1M_QUERY_INIT_MS	This returns the duration of a board initialization in milliseconds.
AI32SSC1M_QUERY_MASTER_CLOCK	This returns the master clock frequency in hertz.
AI32SSC1M_QUERY_NRATE_MASK	This returns the mask for the board's NRATE fields.
AI32SSC1M_QUERY_NRATE_MAX	This returns the maximum supported NRATE value.
AI32SSC1M_QUERY_NRATE_MIN	This returns the minimum supported NRATE value.
AI32SSC1M_QUERY_PRETRIG	This indicates if the Pretrigger feature is or is not present.
AI32SSC1M_QUERY_RATE_GEN_QTY	This returns the number of Rate Generators on the board.
AI32SSC1M_QUERY_V_RANGE	This returns an indicator of the board's voltage range.

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

<b>Value</b>	Description
AI32SSC1M_IOCTL_QUERY_ERROR	Either there was a processing error or the query option is unrecognized.

Valid return values for the voltage range query are as follows.

<b>Value</b>	Description
AI32SSC1M_QUERY_V_RANGE_2_5	The board's input voltage range options are $\pm 2.5$ volts and $\pm 1.25$ volts.
AI32SSC1M_QUERY_V_RANGE_10V	The board's input voltage range options are $\pm 10$ volts and $\pm 5$ volts.

**6.3.36. AI32SSC1M\_IOCTL\_RAG\_ENABLE**

This service enables or disables the Rate-A Generator.

Usage

<b>ioctl( ) Argument</b>	<b>Description</b>
request	AI32SSC1M_IOCTL_RAG_ENABLE
arg	__s32*

Valid argument values are as follows.

<b>Value</b>	<b>Description</b>
-1	Retrieve the current setting.
AI32SSC1M_GEN_ENABLE_NO	This option disables the rate generator.
AI32SSC1M_GEN_ENABLE_YES	This option enables the rate generator.

**6.3.37. AI32SSC1M\_IOCTL\_RAG\_NRATE**

This service configures the NRATE divider value for the Rate-A Generator.

Usage

<b>ioctl( ) Argument</b>	<b>Description</b>
request	AI32SSC1M_IOCTL_RAG_NRATE
arg	__s32*

Valid argument values are from 36 to 0xFFFF.

**6.3.38. AI32SSC1M\_IOCTL\_RBG\_CLK\_SRC**

This service configures the clock source selection for the Rate-B Generator.

Usage

<b>ioctl( ) Argument</b>	<b>Description</b>
request	AI32SSC1M_IOCTL_RBG_CLK_SRC
arg	__s32*

Valid argument values are as follows.

<b>Value</b>	<b>Description</b>
-1	Retrieve the current setting.
AI32SSC1M_RBG_CLK_SRC_MASTER	This refers to the board's master clock.
AI32SSC1M_RBG_CLK_SRC_RAG	This refers to the Rate-A Generator output. This option is used for rate generator cascading.

**6.3.39. AI32SSC1M\_IOCTL\_RBG\_ENABLE**

This service enables or disables the Rate-B Generator.

## Usage

<b>ioctl() Argument</b>	<b>Description</b>
request	AI32SSC1M_IOCTL_RBG_ENABLE
arg	__s32*

Valid argument values are as follows.

<b>Value</b>	<b>Description</b>
-1	Retrieve the current setting.
AI32SSC1M_GEN_ENABLE_NO	This option disables the rate generator.
AI32SSC1M_GEN_ENABLE_YES	This option enables the rate generator.

**6.3.40. AI32SSC1M\_IOCTL\_RBG\_NRATE**

This service configures the NRATE divider value for the Rate-B Generator.

## Usage

<b>ioctl() Argument</b>	<b>Description</b>
request	AI32SSC1M_IOCTL_RBG_NRATE
arg	__s32*

Valid argument values are from 36 to 0xFFFF.

**6.3.41. AI32SSC1M\_IOCTL\_REG\_MOD**

This service performs a read-modify-write of an 18AI32SSC1M register. This includes only the GSC firmware registers. The PCI and PLX Feature Set Registers are read-only. Refer to `18ai32ssc1m.h` for the complete list of GSC firmware registers.

## Usage

<b>ioctl() Argument</b>	<b>Description</b>
request	AI32SSC1M_IOCTL_REG_MOD
arg	gsc_reg_t*

## Definition

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

<b>Fields</b>	<b>Description</b>
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.

**6.3.42. AI32SSC1M\_IOCTL\_REG\_READ**

This service reads the value of an 18AI32SSC1M register. This includes the PCI registers, the PLX Feature Set Registers and the GSC firmware registers. Refer to `18ai32ssc1m.h` and `gsc_pci9056.h` for the complete list of accessible registers.

**Usage**

<b>ioctl() Argument</b>	<b>Description</b>
request	AI32SSC1M_IOCTL_REG_READ
arg	gsc_reg_t*

**Definition**

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

<b>Fields</b>	<b>Description</b>
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.

**6.3.43. AI32SSC1M\_IOCTL\_REG\_WRITE**

This service writes a value to an 18AI32SSC1M register. This includes only the GSC firmware registers. The PCI and PLX Feature Set Registers are read-only. Refer to `18ai32ssc1m.h` for a complete list of the GSC firmware registers.

**Usage**

<b>ioctl() Argument</b>	<b>Description</b>
request	AI32SSC1M_IOCTL_REG_WRITE
arg	gsc_reg_t*

**Definition**

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

<b>Fields</b>	<b>Description</b>
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.

**6.3.44. AI32SSC1M\_IOCTL\_RX\_IO\_ABORT**

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

**Usage**

<b>ioctl()</b> Argument	Description
<code>request</code>	<code>AI32SSC1M_IOCTL_RX_IO_ABORT</code>
<code>arg</code>	<code>__s32*</code>

The results are reported as one of the following values.

<b>Value</b>	Description
<code>AI32SSC1M_IO_ABORT_NO</code>	A <code>read()</code> request was not aborted as none were ongoing.
<code>AI32SSC1M_IO_ABORT_YES</code>	An ongoing <code>read()</code> request was aborted.

**6.3.45. AI32SSC1M\_IOCTL\_RX\_IO\_MODE**

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

**Usage**

<b>ioctl()</b> Argument	Description
<code>request</code>	<code>AI32SSC1M_IOCTL_RX_IO_MODE</code>
<code>arg</code>	<code>__s32*</code>

Valid argument values are as follows.

<b>Value</b>	Description
<code>-1</code>	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. This is the default.

**6.3.46. AI32SSC1M\_IOCTL\_RX\_IO\_OVERFLOW**

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

**NOTE:** The check for an overflow is performed upon entry to the read service. The read service does not check for overflows that occur while the read is in progress. For in-progress overflows an application must perform the check manually or wait for the check performed by a subsequent read request.

**Usage**

<b>ioctl()</b> Argument	Description
<code>request</code>	<code>AI32SSC1M_IOCTL_RX_IO_OVERFLOW</code>
<code>arg</code>	<code>__s32*</code>

Valid argument values are as follows.

<b>Value</b>	Description
<code>-1</code>	Retrieve the current setting.



AI32SSC1M_IO_OVERFLOW_CHECK	Perform the check. This is the default.
AI32SSC1M_IO_OVERFLOW_IGNORE	Do not perform the check.

### 6.3.47. AI32SSC1M\_IOCTL\_RX\_IO\_TIMEOUT

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

#### Usage

ioctl() Argument	Description
request	AI32SSC1M_IOCTL_RX_IO_TIMEOUT
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. The default is 10 seconds.

### 6.3.48. AI32SSC1M\_IOCTL\_RX\_IO\_UNDERFLOW

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

**NOTE:** The check for an underflow is performed upon entry to the read service. The read service does not check for underflows that occur while the read is in progress. For in-progress underflows an application must perform the check manually or wait for the check performed by a subsequent read request.

#### Usage

ioctl() Argument	Description
request	AI32SSC1M_IOCTL_RX_IO_UNDERFLOW
arg	__s32*

Valid argument values are as follows.

Value	Description
-1	Retrieve the current setting.
AI32SSC1M_IO_UNDERFLOW_CHECK	Perform the check. This is the default.
AI32SSC1M_IO_UNDERFLOW_IGNORE	Do not perform the check.

### 6.3.49. AI32SSC1M\_IOCTL\_SAMP\_CLK\_SRC

This service configures the source for the A/D sample clock.

#### Usage

ioctl() Argument	Description
request	AI32SSC1M_IOCTL_SAMP_CLK_SRC
arg	__s32*

Valid argument values are as follows.

Value	Description
-1	Retrieve the current setting.

AI32SSC1M_SAMP_CLK_SRC_BCR	This refers to the Board Control Register's Input Sync bit.
AI32SSC1M_SAMP_CLK_SRC_EXT	This refers to the external clock input signal.
AI32SSC1M_SAMP_CLK_SRC_RAG	This refers to the Rate-A Generator output.
AI32SSC1M_SAMP_CLK_SRC_RBG	This refers to the Rate-B Generator output.

### 6.3.50. AI32SSC1M\_IOCTL\_SCAN\_MARKER

This service configures the insertion of Scan Markers into the input buffer data stream. Refer to the board user manual for additional information.

#### Usage

<b>ioctl() Argument</b>	<b>Description</b>
request	AI32SSC1M_IOCTL_SCAN_MARKER
arg	__s32*

Valid argument values are as follows.

<b>Value</b>	<b>Description</b>
-1	Retrieve the current setting.
AI32SSC1M_SCAN_MARKER_DISABLE	Scan Markers are not inserted into the data stream.
AI32SSC1M_SCAN_MARKER_ENABLE	Scan Markers are inserted into the data stream.

### 6.3.51. AI32SSC1M\_IOCTL\_WAIT\_CANCEL

This service resumes all threads blocked via AI32SSC1M\_IOCTL\_WAIT\_EVENT IOCTL calls (section 6.3.52, page 43), 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

<b>ioctl() Argument</b>	<b>Description</b>
request	AI32SSC1M_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;
```

<b>Fields</b>	<b>Description</b>
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 6.3.52.2 on page 44.
gsc	This specifies the set of AI32SSC1M_WAIT_GSC_* events whose wait requests are to be cancelled. Refer to section 6.3.52.3 on page 44.
alt	This is unused by the 18AI32SSC1M 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 6.3.52.4 on page 44.
timeout_ms	This is unused by wait cancel operations.
count	Upon return this indicates the number of waits that were cancelled.

### 6.3.52. AI32SSC1M\_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

<code>ioctl()</code> Argument	Description
<code>request</code>	<code>AI32SSC1M_IOCTL_WAIT_EVENT</code>
<code>arg</code>	<code>gsc_wait_t*</code>

#### Definition

```
typedef struct
{
    __u32    flags;
    __u32    main;
    __u32    gsc;
    __u32    alt;
    __u32    io;
    __u32    timeout_ms;
    __u32    count;
} gsc_wait_t;
```

Fields	Description
<code>flags</code>	This must initially be zero. Upon return this indicates the reason that the thread was resumed. Refer to section 6.3.52.1 on page 44.
<code>main</code>	This specifies any number of <code>GSC_WAIT_MAIN_*</code> events that the thread is to wait for. Refer to section 6.3.52.2 on page 44.
<code>gsc</code>	This specifies any number of <code>AI32SSC1M_WAIT_GSC_*</code> events that the thread is to wait for. Refer to section 6.3.52.3 on page 44.
<code>alt</code>	This is unused by the 18AI32SSC1M driver and must be zero.
<code>io</code>	This specifies any number of <code>GSC_WAIT_IO_*</code> events that the thread is to wait for. Refer to section 6.3.52.4 on page 44.
<code>timeout_ms</code>	This specified the maximum amount of time, in milliseconds, that the thread is to wait for any of the referenced events. This must be greater than zero. Upon return the value will be the approximate amount of time actually waited.
<code>count</code>	This is unused by wait event operations and must be zero.

6.3.52.1. `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.

6.3.52.2. `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 18AI32SSC1M 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 18AI32SSC1M.
<code>GSC_WAIT_MAIN_PCI</code>	This refers to any interrupt generated by the 18AI32SSC1M.
<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.

6.3.52.3. `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 Interrupt Control Register. Applications are responsible for enabling the desired interrupt options. Refer to `AI32SSC1M_IOCTL_IRQ0_SEL` (section 6.3.30, page 34) and `AI32SSC1M_IOCTL_IRQ1_SEL` (section 6.3.31, page 34).

Value	Description
<code>AI32SSC1M_WAIT_GSC_AUTO_CAL_DONE</code>	This refers to the completion of an auto-calibration cycle.
<code>AI32SSC1M_WAIT_GSC_BURST_DONE</code>	This refers to the completion of an input burst.
<code>AI32SSC1M_WAIT_GSC_BURST_START</code>	This refers to the beginning of an input burst.
<code>AI32SSC1M_WAIT_GSC_IN_BUF_OVR_UNDR</code>	This refers to the occurrence of either an input buffer overflow or an input buffer underflow.
<code>AI32SSC1M_WAIT_GSC_IN_BUF_THR_H2L</code>	This refers to the input buffer threshold status being negated.
<code>AI32SSC1M_WAIT_GSC_IN_BUF_THR_L2H</code>	This refers to the input buffer threshold status being asserted.
<code>AI32SSC1M_WAIT_GSC_INIT_DONE</code>	This refers to the completion of an initialization cycle.
<code>AI32SSC1M_WAIT_GSC_SYNC_DONE</code>	This refers to the completion of a sync operation.
<code>AI32SSC1M_WAIT_GSC_SYNC_START</code>	This refers to the beginning of a sync operation.

6.3.52.4. `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.

GSC_WAIT_IO_RX_TIMEOUT	This refers to read requests which end due to the timeout period lapse.
------------------------	---

### 6.3.53. AI32SSC1M\_IOCTL\_WAIT\_STATUS

This service count all threads blocked via the AI32SSC1M\_IOCTL\_WAIT\_EVENT IOCTL service (section 6.3.52, page 43), 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

<b>ioctl( ) Argument</b>	<b>Description</b>
request	AI32SSC1M_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;
```

<b>Fields</b>	<b>Description</b>
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 6.3.52.2 on page 44.
gsc	This specifies the set of AI32SSC1M_WAIT_GSC_* events whose wait requests are to be counted. Refer to section 6.3.52.3 on page 44.
alt	This is unused by the AI32SSC1M 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 6.3.52.4 on page 44.
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.

## Document History

Revision	Description
December 20, 2011	Updated to release version 1.5.34.0.
November 1, 2011	Updated to release version 1.4.31.0. Various editorial changes. Removed the IRQ_ENABLE, IRQ0_STS and IRQ1_STS IOCTL services. 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.
December 29, 2009	Updated to release version 1.3.13.0.
December 28, 2009	Updated to release version 1.2.13.0.
December 22, 2009	Updated to release version 1.2.12.0. Added Pretrigger services and a few new registers.
October 15, 2009	Updated to release version 1.1.10.0. Added the voltage range query option. Expanded voltage range selection options.
March 27, 2009	Initial driver release.