16AI64SSA/C

64 Channel, 16-Bit Analog Input Board

PMC-16AI64SSA/C
PMC66-16AI64SSA/C
PCIe-16AI64SSA/C

Linux Device Driver
And API Library
User Manual

Manual Revision: August 8, 2023
Driver Release Version 3.12.104.47.0

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

Preface

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

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

Table of Contents

Table of Figures

1. Introduction

1.1. Purpose

The purpose of this document is to describe the interface to the 16AI64SSA API Library and to the underlying Linux device driver. The API Library software provides the interface between "Application Software" and the device driver. The driver software provides the interface between the API Library and the actual 16AI64SSA hardware. The API Library and driver interfaces are based on the board's functionality.

1.2. Acronyms

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

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

1.3. Definitions

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

Term	Definition
…	This is a shortcut representation of the 16AI64SSA installation directory or any of its subdirectories.
16AI64SSA	This is used as a general reference to any device supported by this driver.
16AI64SSA/C	This refers to the 16AI64SSA and 16AI64SSC device families.
API Library	This is a library that provides application-level access to 16AI64SSA hardware.
Application	This is a user mode process, which runs in user space with user mode privileges.
Driver	This is the 16AI64SSA device driver, which runs in kernel space with kernel mode privileges.
Library	This is usually a general reference to the API Library.

1.4. Software Overview

1.4.1. Basic Software Architecture

This section describes the general architecture for the basic components that comprise 16AI64SSA applications. The overall architecture is illustrated in Figure 1 below.

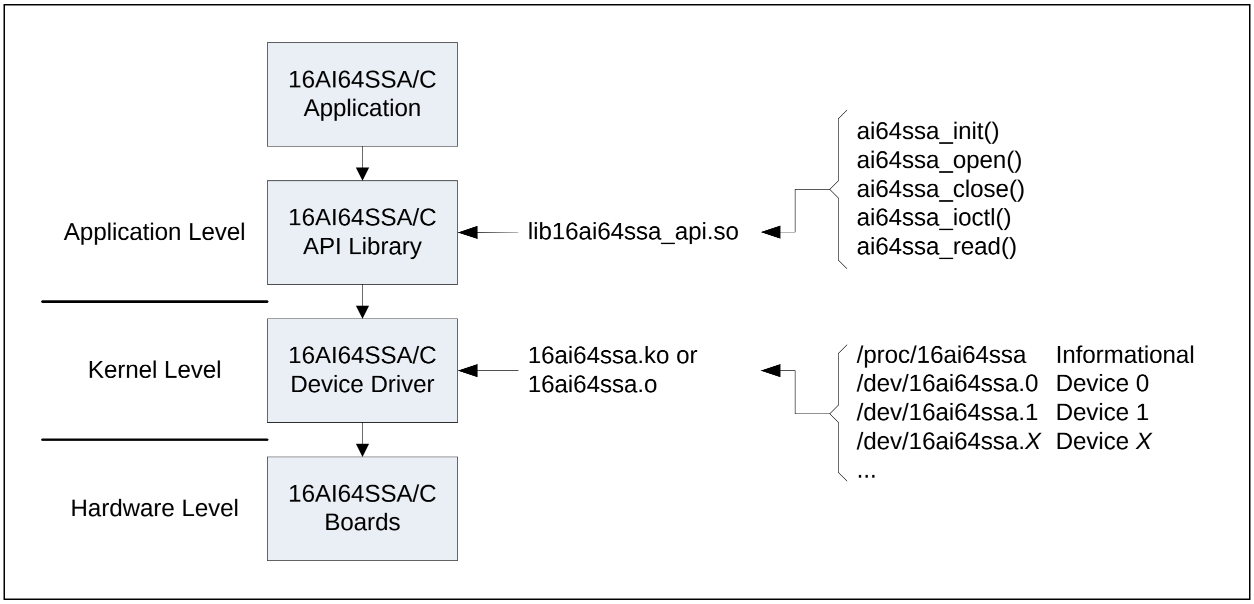

Figure 1 Basic architectural representation.

1.4.2. API Library

The primary means of accessing 16AI64SSA boards is via the 16AI64SSA API Library. This library forms a layer between the application and the driver. Additional information is given in section 4 (page 18). With the library, applications are able to open and close a device and, while open, perform I/O control and read operations.

1.4.3. Device Driver

The device driver is the host software that provides a means of communicating directly with 16AI64SSA hardware. The driver executes under control of the operating system and runs in Kernel Mode as a Kernel Mode device driver. The driver is implemented as a standard dynamically loadable Linux device driver written in the C programming language. While applications can access the driver directly without use of the API Library, it is recommended that all access is made through the library.

1.5. Hardware Overview

The 16AI64SSA is a high-performance 64 channel, 16-bit analog input board. The host side connection is PCI based whose form factor is according to the model ordered. The board is capable of receiving data at up to 200K samples per second over each channel, with an aggregate rate of up to 12.8M samples per second. Internal clocking permits sampling rates from 200K samples per second down to less than one sample per second. Onboard storage permits data buffering of up to 512K samples, for all channels collectively, between the cable interface and the PCI bus. This allows the 16AI64SSA to sustain continuous throughput from the cable interface independent of the PCI bus interface. The 16AI64SSA also permits multiple boards to be synchronized so that all boards sample data in unison. In addition, the board includes autocalibration capability.

1.6. Reference Material

The following reference material may be of particular benefit in using the 16AI64SSA. The specifications provide the information necessary for an in depth understanding of the specialized features implemented on this board.

- The applicable *16AI64SSA User Manual* from General Standards Corporation.
- The applicable *16AI64SSC User Manual* from General Standards Corporation.

- The *PCI9080 PCI Bus Master Interface Chip* data handbook from PLX Technology, Inc.
- The *PCI9056 PCI Bus Master Interface Chip* data handbook from PLX Technology, Inc.

* PLX data books are available from PLX at the following location.

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

1.7. Licensing

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

1.8. Cautionary Notes

1.8.1. IRQ Anomaly

On older firmware, an unintentional interrupt is generated when writing a "1" to an IRQ Request field that is "0". (Refer to the Interrupt Control Register description in the board user manual.) Because of this anomaly, software (including the driver) may miss an interrupt when both IRQ0 and IRQ1 are used and both generate an interrupt almost simultaneously. For applications using IRQ0 and IRQ1 simultaneously it is recommended that firmware with this anomaly not be used. If this is the case with a PLX PCI9080 based board (a PMC-16AI64SSA (or C), for example), then please use firmware version “202” or later. If this is the case with a PLX PCI9056 based board (a PMC66-16AI64SSA (or C), for example), then please use firmware version “222” or later. Refer to the output of the “id” sample application to determine if a board in question is based on the PLX PCI9080 or the PLX PCI9056.

2. Installation

2.1. CPU and Kernel Support

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

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

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

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

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

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

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

2.1.1. 32-bit Support Under 64-bit Environments

This driver supports 32-bit applications under 64-bit environments. The availability of this feature in the kernel depends on a 64-bit kernel being configured to support 32-bit application compatibility. Additionally, 2.6 kernels prior to 2.6.11 implemented 32-bit compatibility in a way that resulted in some drivers not being able to take advantage of the feature. (In these kernels a driver's IOCTL command codes must be globally unique. Beginning with 2.6.11 this requirement has been lifted.) If the driver is not able to provide 32-bit support under a 64-bit kernel, the "`32-bit support`" field in the `/proc/16ai64ssa` file will be "`no`".

2.2. The /proc/ File System

While the driver is running, the text file `/proc/16ai64ssa` can be read to obtain information about the driver and the boards it detects. Each file entry includes an entry name followed immediately by a colon, a space character, and the entry value. Below is an example of what appears in the file, followed by descriptions of each entry.

```
version: 3.12.104.47
32-bit support: yes
boards: 2
models: 16AI64SSA,16AI64SSC
```

Entry	Description
`version`	This gives the driver version number in the form `x.x.x.x`.
`32-bit support`	This reports the driver's support for 32-bit applications. This will be either "`yes`" or "`no`" for 64-bit driver builds and "`yes (native)`" for 32-bit builds.
`boards`	This identifies the total number of boards the driver detected.
`models`	This gives a comma separated list of the basic model number for each board the driver detected. The model numbers are listed in the same order that the boards are accessed via the API Library's open function.

2.3. File List

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

File	Description
`16ai64ssa.linux.tar.gz`	This archive contains the driver, the API Library and all related files.
`16ai64ssa_linux_um.pdf`	This is a PDF version of this user manual, which is included in the archive.

2.4. Directory Structure

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

Directory	Description
`16ai64ssa/`	This is the driver root directory. It contains the documentation, the Overall Make Script (section 2.7, page 13) and the below listed subdirectories.
…`/api/`	This directory contains the API Library source files (section 4, page 18).
…`/docsrc/`	This directory contains the source files for the code samples given in this document (section 6, page 52).
…`/driver/`	This directory contains the device driver source files (section 5, page 48).
…`/include/`	This directory contains the header files for the various libraries.

…/lib/	This directory contains all of the libraries built from the installed sources.
…/samples/	This directory contains the sample application subdirectories and all of their corresponding source files (section 9, page 56).
…/utils/	This directory contains the source files for the utility libraries used by the sample applications (section 7, page 53).

2.5. Installation

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

1. Create and change to the directory where the files are to be installed, such as `/usr/src/linux/drivers/`. (The path name may vary among distributions and kernel versions.)

2. Copy the archive file `16ai64ssa.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 `16ai64ssa` in the current directory, and then copies all of the archive's files into this new directory.

```
tar –xzvf 16ai64ssa.linux.tar.gz
```

2.6. Removal

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

NOTE: The following steps may require elevated privileges.

1. Shutdown the driver as described in section 5.6 (page 51).

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 16ai64ssa.linux.tar.gz 16ai64ssa
```

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

```
rm -f /dev/16ai64ssa.*
```

5. If the automatic startup procedure was adopted (section 5.3.2, page 49), then edit the system startup script `rc.local` and remove the line that invokes the 16AI64SSA's `start` script. The file `rc.local` should be located in the `/etc/rc.d/` directory.

2.7. Overall Make Script

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

NOTE: The following steps may require elevated privileges.

1. Change to the driver root directory (…`/16ai64ssa/`).

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

```
./make_all clean
```

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

```
./make_all
```

2.8. Environment Variables

Some build environments may require compiler or linker options not present in the provided make files. To accommodate local environment specific requirements, the provided make files incorporate support for the following set of GSC specific environment variables.

2.8.1. GSC_API_COMP_FLAGS

This environment variable accommodates adding compiler command line options when compiling source files for the API Library. The compiler used by the API Library make file is "gcc". The content of this environment variable is noted in the make file's output to the screen. The table below shows a portion of the screen output. The "xxx" in the table refers to the contents of the environment variable. This environment variable has no effect on compiling any other distributed source files or linking of any object files.

Undefined or Empty	== Compiling: init.c == Compiling: ioctl.c == Compiling: open.c
Defined and Not Empty	== Compiling: init.c (added 'xxx') == Compiling: ioctl.c (added 'xxx') == Compiling: open.c (added 'xxx')

2.8.2. GSC_API_LINK_FLAGS

This environment variable accommodates adding linker command line options when linking object files for the API Library. The linker used by the API Library make file is "ld". The content of this environment variable is noted in the make file's output to the screen. The table below shows a portion of the screen output. The "xxx" in the table refers to the contents of the environment variable. This environment variable has no effect on compiling of any source files or linking of any other object files.

Undefined or Empty	==== Linking: ../lib/lib16ai64ssa_api.so
Defined and Not Empty	==== Linking: ../lib/lib16ai64ssa_api.so (added 'xxx')

2.8.3. GSC_LIB_COMP_FLAGS

This environment variable accommodates adding compiler command line options when compiling source files for the utility libraries. The compiler used by the utility library make files is "gcc". The content of this environment variable is noted in the make files' output to the screen. The table below shows a portion of the screen output. The "xxx" in the table refers to the contents of the environment variable. This environment variable has no effect on compiling any other distributed source files or linking of any object files.

Undefined or Empty	== Compiling: close.c == Compiling: init.c == Compiling: ioctl.c

Defined and Not Empty	`== Compiling: close.c   (added 'xxx')`
	`== Compiling: init.c   (added 'xxx')`
	`== Compiling: ioctl.c   (added 'xxx')`

2.8.4. GSC_LIB_LINK_FLAGS

This environment variable accommodates adding linker command line options when linking object files for the utility libraries. The linker used by the utility library make files is "ld". The content of this environment variable is noted in the make files' output to the screen. The table below shows a portion of the screen output. The "xxx" in the table refers to the contents of the environment variable. This environment variable has no effect on compiling of any source files or linking of any other object files.

Undefined or Empty	`==== Linking: ../lib/16ai64ssa_utils.a`
Defined and Not Empty	`==== Linking: ../lib/16ai64ssa_utils.a   (added 'xxx')`

2.8.5. GSC_APP_COMP_FLAGS

This environment variable accommodates adding compiler command line options when compiling source files for the sample applications. The compiler used by the sample application make files is "gcc". The content of this environment variable is noted in the make files' output to the screen. The table below shows a portion of the screen output. The "xxx" in the table refers to the contents of the environment variable. This environment variable has no effect on compiling any other distributed source files or linking of any object files.

Undefined or Empty	`== Compiling: main.c` `== Compiling: perform.c`
Defined and Not Empty	`== Compiling: main.c   (added 'xxx')` `== Compiling: perform.c   (added 'xxx')`

2.8.6. GSC_APP_LINK_FLAGS

This environment variable accommodates adding linker command line options when linking object files for the sample applications. The linker used by the sample application make files is "gcc". The content of this environment variable is noted in the make files' output to the screen. The table below shows a portion of the screen output. The "xxx" in the table refers to the contents of the environment variable. This environment variable has no effect on compiling of any source files or linking of any other object files.

Undefined or Empty	`==== Linking: id`
Defined and Not Empty	`==== Linking: id   (added 'xxx')`

3. Main Interface Files

This section gives general information on the suggested device interface files to use when developing 16AI64SSA based applications.

3.1. Main Header File

Throughout the remainder of this document references are made to various header files included as part of the 16AI64SSA driver installation. For ease of use it is suggested that applications include only the single header file shown below rather than individually including those headers identified separately later in this document. Including this header file pulls in all other pertinent 16AI64SSA specific header files. Therefore, sources may include only this one 16AI64SSA header and make files may reference only this one 16AI64SSA include directory.

Description	File	Location
Header File	`16ai64ssa_main.h`	`…/include/`

3.2. Main Library File

Throughout the remainder of this document references are made to various statically linkable libraries included as part of the 16AI64SSA driver installation. For ease of use it is suggested that applications link only the single library file shown below rather than individually linking those libraries identified separately later in this document. Linking this library file pulls in all other static libraries included with the driver. Therefore, make files may reference only this one 16AI64SSA static library and only this one 16AI64SSA library directory.

Description	File	Location
Static Library	`16ai64ssa_main.a`	`…/lib/`
	`16ai64ssa_multi.a`	

> **NOTE**: For applications using the 16AI64SSA and no other GSC devices, link the `16ai64ssa_main.a` library. For applications using multiple GSC device types, link the `xxxx_main.a` library for one of the devices and the `xxxx_multi.a` library for the others. Linking multiple `xxxx_main.a` libraries may likely produce link errors due to duplicate symbols being defined. While it may make little or no difference, it is recommended that one choose the `xxxx_main.a` library from the driver with the largest number in positions three (x.x.X.x.x) and/or four (x.x.x.X.x) in the driver release version number.

> **NOTE**: The 16AI64SSA API Library is implemented as a shared library and is thus not linked with the 16AI64SSA Main Library. The API Library must be linked with applications by adding the argument `–l16ai64ssa_api` to the linker command line.

3.2.1. Build

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

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

2. Remove existing build targets using the below command.

   ```
   make clean
   ```

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

   ```
   make
   ```

3.2.2. System Libraries

In addition to linking the static library named above, as well as the API Library shared object file, applications may need to also link in additional system libraries as noted below.

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

4. API Library

The 16AI64SSA API Library is the software interface between user applications and the 16AI64SSA device driver. The interface is accessed by including the header file 16ai64ssa_api.h.

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

4.1. Files

The library files are summarized in the table below.

Description	File	Location
Source Files	*.c, *.h …	…/api/
Header File	16ai64ssa_api.h	…/include/
Library File	lib16ai64ssa_api.so	…/lib/ /usr/lib/ †

† The shared object library is automatically copied to /usr/lib/ when it is built.

4.2. Build

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

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

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

2. Remove existing build targets using the below command.

   ```
   make clean
   ```

3. Compile the source files and build the library by issuing the below command. This step copies the API Library file to /usr/lib/.

   ```
   make
   ```

4.3. Library Use

The library is used at application compile time, at application link time and at application run time. At compile time include the below listed header file in each source file using a component of the Library interface. Also, edit the include file search path to locate the header file in the below listed directory. At link time the Library's shared object file is linked via the linker command line. This can be done by naming the .so file explicitly or by adding the below linker command line argument. At run time the library is found in the directory /usr/lib/. (The shared object file is automatically copied to /usr/lib/ when it is built.)

Description	File	Location	Linker Argument
Header File	16ai64ssa_api.h	…/include/	
Shared Object Library	lib16ai64ssa_api.so	…/lib/	
		/usr/lib/	-l16ai64ssa_api

4.4. Macros

The API Library and driver interfaces include the following macros, which are defined in 16ai64ssa.h.

4.4.1. IOCTL Services

The IOCTL macros are documented in section 4.7 (page 24).

4.4.2. Registers

The following gives the complete set of 16AI64SSA registers.

4.4.2.1. GSC Registers

The following table gives the complete set of GSC specific 16AI64SSA registers. Please note that the set of registers supported by any given device may vary according to model and firmware version. For the set of supported registers and their detailed definitions refer to the appropriate *16AI64SSA User Manual*.

> **NOTE**: Refer to the output of the “id” sample application (`…/id/`) for a complete list of the registers supported by the device being accessed.

Macro	Description
AI64SSA_GSC_ACAR	Active Channel Assignment Register (ACAR)
AI64SSA_GSC_ARWR	Auxiliary R/W Register (ARWR)
AI64SSA_GSC_ASIOCR	Auxiliary Sync I/O Control Register (ASIOCR)
AI64SSA_GSC_AVR	Autocal Values Register (AVR)
AI64SSA_GSC_BCFGR	Board Configuration Register (BCFGR)
AI64SSA_GSC_BCTLR	Board Control Register (BCTLR)
AI64SSA_GSC_BUFSR	Buffer Size Register (BUFSR)
AI64SSA_GSC_BURSR	Burst Size Register (BURSR)
AI64SSA_GSC_IBCR	Input Buffer Control Register (IBCR)
AI64SSA_GSC_ICR	Interrupt Control Register (ICR)
AI64SSA_GSC_IDBR	Input Data Buffer Register (IDBR)
AI64SSA_GSC_LLCR	Low Latency Control Register (LLCR) *
AI64SSA_GSC_LLHRXX	Low Latency Holding Registers 00 to 63 (LLHR00-LLHR63) *
AI64SSA_GSC_RAGR	Rate-A Generator Register (RAGR)
AI64SSA_GSC_RBGR	Rate-B Generator Register (RBGR)
AI64SSA_GSC_SMLWR	Scan Marker Lower Word Register (SMLWR)
AI64SSA_GSC_SMUWR	Scan Marker Upper Word Register (SMUWR)
AI64SSA_GSC_SSCR	Scan & Sync Control Register (SSCR)

* Available only if supported in firmware.

4.4.2.2. PCI Configuration Registers

Access to the PCI registers is seldom required so these registers are not listed here. For the complete list of the PCI register identifiers refer to the driver header files `gsc_pci9080.h` and `gsc_pci9056.h`, which are automatically included via `16ai64ssa_api.h`.

4.4.2.3. PLX Feature Set Registers

Access to the PLX registers is seldom required so these registers are not listed here. For the complete list of the PLX register identifiers refer to the driver header files `gsc_pci9080.h` and `gsc_pci9056.h`, which are automatically included via `16ai64ssa_api.h`.

4.5. Data Types

The data types used by the API Library are described with the IOCTL services with which they are used. For additional information refer to section 4.7 (page 24).

4.6. Functions

The interface includes the following functions. The return values reflect the completion status of the requested operation. A return value less than zero always reflects an error condition. The table below summarizes the error status values. For the I/O function, read, non-negative return values reflect the number of bytes transferred between the application and the interface. A value equal to the requested transfer size indicates complete success. Return values less than the requested transfer size indicate that the I/O timeout expired. For the other API function calls a return value of zero indicates success.

Return Value	Description
`< 0`	This is the value "`(-errno)`" (see `errno.h`).

4.6.1. ai64ssa_close()

This function is the entry point to close a connection made via the API's open call (section 4.6.4, page 22). The device is put in an initialized state before this call returns.

Prototype

```
int ai64ssa_close(int fd);
```

Argument	Description
`fd`	This is the file descriptor obtained from the open service (section 4.6.4, page 22).

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

Example

```
#include <stdio.h>

#include "16ai64ssa_dsl.h"

int ai64ssa_close_dsl(int fd)
{
    int errs;
    int ret;

    ret = ai64ssa_close(fd);

    if (ret)
        printf("ERROR: ai64ssa_close() returned %d\n", ret);

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

4.6.2. ai64ssa_init()

This function is the entry point to initializing the 16AI64SSA API Library and must be the first call into the Library. This function may be called more than once, but only the first successful call actually initializes the library. Subsequent calls perform no operation at all. All other API calls return a failure status when the API Library is not initialized.

Prototype

```
int ai64ssa_init(void);
```

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

Example

```
#include <stdio.h>

#include "16ai64ssa_dsl.h"

int ai64ssa_init_dsl(void)
{
    int errs;
    int ret;

    ret = ai64ssa_init();

    if (ret)
        printf("ERROR: ai64ssa_init() returned %d\n", ret);

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

4.6.3. ai64ssa_ioctl()

This function is the entry point to performing setup and control operations on a 16AI64SSA. This function should only be called after a successful open of the respective device. The specific operation performed varies according to the `request` argument. The `request` argument also governs the use and interpretation of the `arg` argument. The set of supported options for the `request` argument consists of the IOCTL services supported by the driver, which are defined in section 4.7 (page 24).

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

Prototype

```
int ai64ssa_ioctl(int fd, int request, void* arg);
```

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

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

Example

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

```
#include "16ai64ssa_dsl.h"

int ai64ssa_ioctl_dsl(int fd, int request, void* arg)
{
    int errs;
    int ret;

    ret = ai64ssa_ioctl(fd, request, arg);

    if (ret)
        printf("ERROR: ai64ssa_ioctl() returned %d\n", ret);

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

4.6.4. ai64ssa_open()

This function is the entry point to open a connection to a 16AI64SSA board. Before returning, the initialize IOCTL service is called to reset all hardware and software settings to their defaults.

Prototype

```
int ai64ssa_open(int device, int share, int* fd);
```

<table>
<tr><th>Argument</th><th>Description</th></tr>
<tr><td><code>device</code></td><td>This is the zero-based index of the 16AI64SSA to access. †</td></tr>
<tr><td><code>share</code></td><td>Open the device in Shared Access Mode? If non-zero the device is opened in Shared Access Mode (see below). If zero the device is opened in Exclusive Access Mode (see below).</td></tr>
<tr><td><code>fd</code></td><td>The device handle is returned here. The pointer cannot be NULL. Values returned are as follows.
<table>
<tr><th>Value</th><th>Description</th></tr>
<tr><td><code>>= 0</code></td><td>This is the handle to use to access the device in subsequent calls.</td></tr>
<tr><td><code>-1</code></td><td>There was an error. The device is not accessible.</td></tr>
</table>
</td></tr>
</table>

† The index value `-1` can also be given to acquire driver information (section 2.2, page 12).

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

Example

```
#include <stdio.h>

#include "16ai64ssa_dsl.h"

int ai64ssa_open_dsl(int device, int share, int* fd)
{
    int errs;
    int ret;

    ret = ai64ssa_open(device, share, fd);
```

```
    if (ret)
        printf("ERROR: ai64ssa_open() returned %d\n", ret);

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

4.6.4.1. Access Modes

The value of the `share` argument determines the device access mode, as follows.

Shared Access Mode:

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

Exclusive Access Mode:

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

4.6.5. ai64ssa_read()

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

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

> **NOTE:** For additional information refer to the Data Transfer Modes section (section 8.3, page 54).

Prototype

```
int ai64ssa_read(int fd, void* dst, size_t bytes);
```

Argument	Description
`fd`	This is the file descriptor obtained from the open service (section 4.6.4, page 22).
`dst`	The data read is put here.
`bytes`	This is the desired number of bytes to read. When reading from a device, this must be a multiple of four (4).

Return Value	Description
0 to bytes	The operation succeeded. When reading from a device, a value less than bytes indicates that the I/O timeout period lapsed (section 4.7.48, page 41) before the entire request could be satisfied.
< 0	An error occurred. See error value description above.

Example

```
#include <stdio.h>

#include "16ai64ssa_dsl.h"

int ai64ssa_read_dsl(int fd, void* dst, size_t bytes, size_t* qty)
{
    int errs;
    int ret;

    ret = ai64ssa_read(fd, dst, bytes);

    if (ret < 0)
        printf("ERROR: ai64ssa_read() returned %d\n", ret);

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

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

4.7. IOCTL Services

The 16AI64SSA API Library and device driver implement the following IOCTL services. Each service is described along with the applicable ai64ssa_ioctl() function arguments.

4.7.1. AI64SSA_IOCTL_AI_BUF_CLEAR

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

Usage

Argument	Description
request	AI64SSA_IOCTL_AI_BUF_CLEAR
arg	Not used.

4.7.2. AI64SSA_IOCTL_AI_BUF_LEVEL

This service returns the current number of 32-bit data items in the input buffer.

Usage

Argument	Description
request	AI64SSA_IOCTL_AI_BUF_LEVEL
arg	s32*

The value returned will be from zero up to the size of the input buffer.

4.7.3. AI64SSA_IOCTL_AI_BUF_OVERFLOW

This service operates on the Input Buffer Overflow status.

Usage

Argument	Description
request	AI64SSA_IOCTL_AI_BUF_OVERFLOW
arg	s32*

Valid argument values supplied to the service are as follows.

Value	Description
-1	Retrieve the current state.
AI64SSA_AI_BUF_OVERFLOW_CHECK	Check to see if there has been an overflow.
AI64SSA_AI_BUF_OVERFLOW_CLEAR	Clear the overflow status.

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

Value	Description
AI64SSA_AI_BUF_OVERFLOW_NO	The buffer has not experienced an overflow condition.
AI64SSA_AI_BUF_OVERFLOW_YES	The buffer has experienced an overflow condition.

4.7.4. AI64SSA_IOCTL_AI_BUF_THR_LVL

This service configures the input buffer threshold level.

Usage

Argument	Description
request	AI64SSA_IOCTL_AI_BUF_THR_LVL
arg	s32*

Valid argument values are from zero up to the size of the input buffer, and -1. A value of -1 will return the current threshold level setting.

> **NOTE:** On models with a prefix of “PMC-” the threshold level is rounded to the nearest lower multiple of four when set to 64K or higher (more precisely, 0xFFFF or higher). On other models (i.e., “PMC66-” and “PCIe-”) no such rounding occurs.

4.7.5. AI64SSA_IOCTL_AI_BUF_THR_STS

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

Usage

Argument	Description
request	AI64SSA_IOCTL_AI_BUF_THR_STS
arg	s32*

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

Value	Description
AI64SSA_AI_BUF_THR_STS_CLEAR	The input buffer contains Threshold Level number of data items, or fewer.
AI64SSA_AI_BUF_THR_STS_SET	The input buffer contains more than Threshold Level number of data items.

4.7.6. AI64SSA_IOCTL_AI_BUF_UNDERFLOW

This service operates on the Input Buffer Underflow status.

Usage

Argument	Description
request	AI64SSA_IOCTL_AI_BUF_UNDERFLOW
arg	s32*

Valid argument values supplied to the service are as follows.

Value	Description
-1	Retrieve the current state.
AI64SSA_AI_BUF_UNDERFLOW_CHECK	Check to see if there has been an underflow.
AI64SSA_AI_BUF_UNDERFLOW_CLEAR	Clear the underflow status.

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

Value	Description
AI64SSA_AI_BUF_UNDERFLOW_NO	The buffer has not experienced an underflow condition.
AI64SSA_AI_BUF_UNDERFLOW_YES	The buffer has experienced an underflow condition.

4.7.7. AI64SSA_IOCTL_AI_MODE

This service configures the board’s Analog Input Mode.

Usage

Argument	Description
request	AI64SSA_IOCTL_AI_MODE
arg	s32*

Valid argument values are as follows.

Value	Description
-1	Retrieve the current setting.
AI64SSA_AI_MODE_DIFF	Configure the input channels for full differential operation. *
AI64SSA_AI_MODE_PS_DIFF	Configure the input channels for pseudo-differential operation. *
AI64SSA_AI_MODE_SINGLE	Configure the input channels for single-ended operation.

AI64SSA_AI_MODE_VREF	Configure the input channels for +VREF input testing
AI64SSA_AI_MODE_ZERO	Configure the input channels for Zero input testing

* The differential input modes are not compatible with the unipolar voltage ranges (section 4.7.8, page 27).

4.7.8. AI64SSA_IOCTL_AI_RANGE

This service configures the analog input voltage range.

Usage

Argument	Description
request	AI64SSA_IOCTL_AI_RANGE
arg	s32*

Valid argument values are as follows.

Value	Description
-1	Retrieve the current setting.
AI64SSA_AI_RANGE_0_5V	Set the input voltage range to 0 to 5 volts. *
AI64SSA_AI_RANGE_0_10V	Set the input voltage range to 0 to 10 volts. *
AI64SSA_AI_RANGE_2_5V	Set the input voltage range to ±2.5 volts.
AI64SSA_AI_RANGE_5V	Set the input voltage range to ±5 volt.
AI64SSA_AI_RANGE_10V	Set the input voltage range to ±10 volts.

* The unipolar voltage ranges are not compatible with the differential input modes (section 4.7.7, page 26).

4.7.9. AI64SSA_IOCTL_AUTOCAL

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

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

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

Usage

Argument	Description
request	AI64SSA_IOCTL_AUTOCAL
arg	Not used.

4.7.10. AI64SSA_IOCTL_AUTOCAL_STATUS

This service returns the status of the most recent Autocalibration cycle.

Usage

Argument	Description
request	AI64SSA_IOCTL_AUTOCAL_STATUS
arg	s32*

Argument values returned are as follows.

Value	Description
AI64SSA_AUTOCAL_STATUS_ACTIVE	The operation is still in progress.
AI64SSA_AUTOCAL_STATUS_FAIL	The operation failed.
AI64SSA_AUTOCAL_STATUS_PASS	The operation passed.

4.7.11. AI64SSA_IOCTL_AUX_0_MODE

This service configures the Auxiliary 0 cable interface signal.

Usage

Argument	Description
request	AI64SSA_IOCTL_AUX_0_MODE
arg	s32*

Valid argument values are as follows.

Value	Description
-1	Retrieve the current setting.
AI64SSA_AUX_MODE_DISABLE	This option disables the signal.
AI64SSA_AUX_MODE_INPUT	This option configures the signal as an input.
AI64SSA_AUX_MODE_OUTPUT	This option configures the signal as an output.

4.7.12. AI64SSA_IOCTL_AUX_1_MODE

This service configures the Auxiliary 1 cable interface signal.

Usage

Argument	Description
request	AI64SSA_IOCTL_AUX_0_MODE
arg	s32*

Valid argument values are as follows.

Value	Description
-1	Retrieve the current setting.
AI64SSA_AUX_MODE_DISABLE	This option disables the signal.
AI64SSA_AUX_MODE_INPUT	This option configures the signal as an input.
AI64SSA_AUX_MODE_OUTPUT	This option configures the signal as an output.

4.7.13. AI64SSA_IOCTL_AUX_2_MODE

This service configures the Auxiliary 0 cable interface signal.

Usage

Argument	Description
request	AI64SSA_IOCTL_AUX_2_MODE
arg	s32*

Valid argument values are as follows.

Value	Description
-1	Retrieve the current setting.
AI64SSA_AUX_MODE_DISABLE	This option disables the signal.
AI64SSA_AUX_MODE_INPUT	This option configures the signal as an input.
AI64SSA_AUX_MODE_OUTPUT	This option configures the signal as an output.

4.7.14. AI64SSA_IOCTL_AUX_3_MODE

This service configures the Auxiliary 3 cable interface signal.

Usage

Argument	Description
request	AI64SSA_IOCTL_AUX_0_MODE
arg	s32*

Valid argument values are as follows.

Value	Description
-1	Retrieve the current setting.
AI64SSA_AUX_MODE_DISABLE	This option disables the signal.
AI64SSA_AUX_MODE_INPUT	This option configures the signal as an input.
AI64SSA_AUX_MODE_OUTPUT	This option configures the signal as an output.

4.7.15. AI64SSA_IOCTL_AUX_IN_POL

This service configures the polarity of the auxiliary cable signals configured as inputs.

Usage

Argument	Description
request	AI64SSA_IOCTL_AUX_IN_POL
arg	s32*

Valid argument values are as follows.

Value	Description
-1	Retrieve the current setting.
AI64SSA_AUX_IN_POL_HI_2_LO	This option configures the signals for High to Low operation.
AI64SSA_AUX_IN_POL_LO_2_HI	This option configures the signals for Low to High operation.

4.7.16. AI64SSA_IOCTL_AUX_NOISE

This service configures the auxiliary cable signals for noise sensitivity.

Usage

Argument	Description
request	AI64SSA_IOCTL_AUX_NOISE
arg	s32*

Valid argument values are as follows.

Value	Description
-1	Retrieve the current setting.
AI64SSA_AUX_NOISE_LOW	This option configures the signals for low noise environments.
AI64SSA_AUX_NOISE_HIGH	This option configures the signals for high noise environments.

4.7.17. AI64SSA_IOCTL_AUX_OUT_POL

This service configures the polarity of the auxiliary cable signals configured as outputs.

Usage

Argument	Description
request	AI64SSA_IOCTL_AUX_OUT_POL
arg	s32*

Valid argument values are as follows.

Value	Description
-1	Retrieve the current setting.
AI64SSA_AUX_OUT_POL_HI_PULSE	This option configures the signals for high going pulse generation.
AI64SSA_AUX_OUT_POL_LOW_PULSE	This option configures the signals for low going pulse generation.

4.7.18. AI64SSA_IOCTL_BURST_SIZE

This service configures the size of a single input burst. The count is in scans, which is a complete A/D conversion of all active input channels.

Usage

Argument	Description
request	AI64SSA_IOCTL_AI_BURST_SIZE
arg	s32*

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

4.7.19. AI64SSA_IOCTL_BURST_SRC

This service enables and disables input bursting and selects the trigger source when bursting is enabled.

Usage

Argument	Description
request	AI64SSA_IOCTL_BURST_SRC
arg	s32*

Valid argument values are as follows.

Value	Description
-1	Retrieve the current setting.
AI64SSA_BURST_SRC_BCR	This option enables bursting and sets the trigger source as software initiated.
AI64SSA_BURST_SRC_DISABLE	This option disables Burst Mode operation.

AI64SSA_BURST_SRC_EXT	This option enables bursting and sets the trigger source as the External Sync Input.
AI64SSA_BURST_SRC_RBG	This option enables bursting and sets the trigger source as the Rate-B Generator.

4.7.20. AI64SSA_IOCTL_BURST_STATUS

This service reports on the board's input burst status.

Usage

Argument	Description
request	AI64SSA_IOCTL_BURST_STATUS
arg	s32*

The value returned will be one of the following.

Value	Description
AI64SSA_BURST_STATUS_ACTIVE	The board is performing a burst operation.
AI64SSA_BURST_STATUS_IDLE	The board is not performing a burst operation.

4.7.21. AI64SSA_IOCTL_CHAN_ACTIVE

This service sets the range and number of active input channels.

Usage

Argument	Description
request	AI64SSA_IOCTL_CHAN_ACTIVE
arg	s32*

Valid argument values are as follows.

Value	Description
-1	Retrieve the current setting.
AI64SSA_CHAN_ACTIVE_0_1	This sets the active channel range to zero through one.
AI64SSA_CHAN_ACTIVE_0_3	This sets the active channel range to zero through three.
AI64SSA_CHAN_ACTIVE_0_7	This sets the active channel range to zero through seven.
AI64SSA_CHAN_ACTIVE_0_15	This sets the active channel range to zero through 15.
AI64SSA_CHAN_ACTIVE_0_31	This sets the active channel range to zero through 31.
AI64SSA_CHAN_ACTIVE_0_63	This sets the active channel range to zero through 63. *
AI64SSA_CHAN_ACTIVE_RANGE	This sets the active channels to a user specified range.
AI64SSA_CHAN_ACTIVE_SINGLE	This sets the active channel to a single user specified channel.

* This option is available only on boards which have 64 channels installed.

4.7.22. AI64SSA_IOCTL_CHAN_FIRST

This service sets the first channel number to scan when the active channel option is set to the Range option (see AI64SSA_CHAN_ACTIVE_RANGE, section 4.7.21, page 31).

Usage

Argument	Description
request	AI64SSA_IOCTL_CHAN_FIRST
arg	s32*

Valid argument values are from zero the one less than the number of available channels, and -1. A value of -1 will return the current setting. If the setting specified is greater than the current *last* setting, then the *last* setting is silently set to equal the specified *first* setting.

4.7.23. AI64SSA_IOCTL_CHAN_LAST

This service sets the last channel number to scan when the active channel option is set to the Range option (see AI64SSA_CHAN_ACTIVE_RANGE, section 4.7.21, page 31).

Usage

Argument	Description
request	AI64SSA_IOCTL_CHAN_LAST
arg	s32*

Valid argument values are from zero the one less than the number of available channels, and -1. A value of -1 will return the current setting. If the setting specified is less than the current *first* setting, then the *first* setting is silently set to equal the specified *last* setting.

4.7.24. AI64SSA_IOCTL_CHAN_SINGLE

This service sets the channel number to scan when the active channel option is set to the Single option (see AI64SSA_IOCTL_BURST_SRC, section 4.7.19, page 30).

Usage

Argument	Description
request	AI64SSA_IOCTL_CHAN_SINGLE
arg	s32*

Valid argument values are from zero the one less than the number of available channels, and -1. A value of -1 will return the current setting.

4.7.25. AI64SSA_IOCTL_DATA_FORMAT

This service sets the data encoding format.

Usage

Argument	Description
request	AI64SSA_IOCTL_DATA_FORMAT
arg	s32*

Valid argument values are as follows.

Value	Description
-1	Retrieve the current setting.

AI64SSA_DATA_FORMAT_2S_COMP	Select the Twos Compliment data format.
AI64SSA_DATA_FORMAT_OFF_BIN	Select the Offset Binary encoding format.

4.7.26. AI64SSA_IOCTL_DATA_PACKING

This service enables or disables data packing.

> **NOTE:** Data Packing can be enabled only on boards that include the Data Packing feature in firmware (see Query option AI64SSA_QUERY_DATA_PACKING, section 4.7.36, page 36).

Usage

Argument	Description
request	AI64SSA_IOCTL_DATA_PACKING
arg	s32*

Valid argument values are as follows.

Value	Description
-1	Retrieve the current setting.
AI64SSA_DATA_PACKING_DISABLE	This refers to Data Packing being disabled.
AI64SSA_DATA_PACKING_ENABLE	This refers to Data Packing being enabled.

4.7.27. AI64SSA_IOCTL_EXT_SYNC_ENABLE

This service enables or disables external synchronization for configurations synchronizing sampling multiple boards.

Usage

Argument	Description
request	AI64SSA_IOCTL_EXT_SYNC_ENABLE
arg	s32*

Valid argument values are as follows.

Value	Description
-1	Retrieve the current setting.
AI64SSA_EXT_SYNC_ENABLE_NO	This option disables external synchronization.
AI64SSA_EXT_SYNC_ENABLE_YES	This option enables external synchronization.

4.7.28. AI64SSA_IOCTL_INITIALIZE

This service resets all hardware and software settings to their defaults.

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

Usage

Argument	Description
command	AI64SSA_IOCTL_INITIALIZE
arg	None

4.7.29. AI64SSA_IOCTL_INPUT_SYNC

This service initiates an Input Sync operation. The driver will wait for completion, but no more than the read timeout period (though not the infinite option). If the read timeout is zero, then the driver will wait up to one second for completion. (See AI64SSA_IOCTL_RX_IO_TIMEOUT, section 4.7.48, page 41.)

Usage

Argument	Description
request	AI64SSA_IOCTL_INPUT_SYNC
arg	Not used.

4.7.30. AI64SSA_IOCTL_IRQ0_SEL

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

NOTE: IRQ Anomaly: Please refer to the Licensing
For licensing information please refer to the text file LICENSE.txt in the root installation directory.
Cautionary Notes about an IRQ Anomaly present in older formware (section 1.8.1, page 10).

Usage

Argument	Description
request	AI64SSA_IOCTL_IRQ0_SEL
arg	s32*

Valid argument values are as follows.

Value	Description
-1	Retrieve the current setting.
AI64SSA_IRQ0_AUTOCAL_DONE	This refers to the completion of an Autocalibration cycle.
AI64SSA_IRQ0_INIT_DONE	This refers to the completion of an initialization cycle.
AI64SSA_IRQ0_SYNC_START	This refers to the beginning of an input scan operation.
AI64SSA_IRQ0_SYNC_DONE	This refers to the completion of an input scan operation.
AI64SSA_IRQ0_BURST_DONE	This refers to the completion of an input burst operation.
AI64SSA_IRQ0_BURST_START	This refers to the beginning of an input burst operation.

4.7.31. AI64SSA_IOCTL_IRQ1_SEL

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

NOTE: IRQ Anomaly: Please refer to the Licensing
For licensing information please refer to the text file LICENSE.txt in the root installation directory.
Cautionary Notes about an IRQ Anomaly present in older formware (section 1.8.1, page 10).

Usage

Argument	Description
request	AI64SSA_IOCTL_IRQ1_SEL
arg	s32*

Valid argument values are as follows.

Value	Description
-1	Retrieve the current setting.
AI64SSA_IRQ1_NONE	This disabled the interrupt.
AI64SSA_IRQ1_IN_BUF_THR_H2L	This refers to the input buffer threshold status being negated.
AI64SSA_IRQ1_IN_BUF_THR_L2H	This refers to the input buffer threshold status being asserted.
AI64SSA_IRQ1_BUF_ERROR	This refers to the input buffer overflow or underflow.

4.7.32. AI64SSA_IOCTL_LOW_LAT_DOH

This service reports the Data On Hold status for boards with the Low Latency feature.

Usage

Argument	Description
request	AI64SSA_IOCTL_LOW_LAT_DOH
arg	s32*

Valid argument values are as follows.

Value	Description
AI64SSA_LOW_LAT_DOH_NO	Low Latency data is not being retrieved.
AI64SSA_LOW_LAT_DOH_YES	Low Latency data is being retrieved.

4.7.33. AI64SSA_IOCTL_LOW_LAT_HOLD_CHAN

This service sets the Low Latency HOLD Channel number.

Usage

Argument	Description
request	AI64SSA_IOCTL_LOW_LAT_HOLD_CHAN
arg	s32*

Valid argument values are in the range from zero to 63 for 64 channel boards, from zero to 31 for 32 channel boards, and -1. A value of -1 is used to retrieve the current setting. The default is 63 for 64 channel boards and 31 for 32 channel boards. If the Low Latency feature is supported but the Low Latency Control Register is not, then the HOLD channel number is fixed at channel zero.

4.7.34. AI64SSA_IOCTL_LOW_LAT_READ

This service performs a read of the board's Low Latency Hold Registers. All hold registers are read sequentially from lowest to highest with the value obtained returned to the caller, except as follows. If the HOLD or RELEASE channel numbers are invalid, then all channels are returned as zero and are not read. If the HOLD and RELEASE channel numbers are equal, then that register is read while all others are returned as zero. If the HOLD channel number is less than the RELEASE channel number, then channels outside that range are returned as zero and are not read. If the board is configured for Full Differential operation, then the odd channels are returned as zero and are not read. On 32 channel boards the upper 32 channels (32 through 63) are returned as zero and are not read. The HOLD channel register is always read first. The RELEASE channel register is always read last. All values read from the hold registers are returned to the caller. All others are returned as zero.

Usage

Argument	Description
request	AI64SSA_IOCTL_LOW_LAT_READ
arg	ai64ssa_ll_t*

Definition

```
typedef struct
{
    u16 data[64];
} ai64ssa_ll_t;
```

Fields	Description
Data	The data read from the Hold Registers is placed here.

4.7.35. AI64SSA_IOCTL_LOW_LAT_REL_CHAN

This service sets the Low Latency RELEASE Channel number.

Usage

Argument	Description
request	AI64SSA_IOCTL_LOW_LAT_HOLD_CHAN
arg	s32*

Valid argument values are in the range from zero to 63 for 64 channel boards, from zero to 31 for 32 channel boards, and -1. A value of -1 is used to retrieve the current setting. The default is 63 for 64 channel boards and 31 for 32 channel boards. If the Low Latency feature is supported but the Low Latency Control Register is not, then the RELEASE channel number is fixed at 63 for 64 channel boards and 31 for 32 channel boards.

4.7.36. AI64SSA_IOCTL_QUERY

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

Usage

Argument	Description
request	AI64SSA_IOCTL_QUERY
arg	s32*

Valid argument values are as follows.

Value	Description
AI64SSA_QUERY_AUTOCAL_MS	This returns the maximum duration of the Autocalibration cycle in milliseconds.
AI64SSA_QUERY_CHAN_RANGE	This reports if the “Range” channel selection option is supported.

AI64SSA_QUERY_CHANNEL_MAX	This returns the maximum number of input channels supported by the board, which may be more that the board's current configuration.
AI64SSA_QUERY_CHANNEL_QTY	This returns the actual number of input channels on the current board.
AI64SSA_QUERY_COUNT	This returns the number of query options supported by the IOCTL service.
AI64SSA_QUERY_DATA_PACKING	This reports if the board supports the Data Packing feature.
AI64SSA_QUERY_DEVICE_TYPE	This returns the identifier value for the board's type. This should be either GSC_DEV_TYPE_16AI64SSA or GSC_DEV_TYPE_16AI64SSC.
AI64SSA_QUERY_FIFO_SIZE	This returns the size of the input buffer in 32-bit A/D values.
AI64SSA_QUERY_FSAMP_MAX	This returns the maximum sample rate, FSAMP, in samples per second.
AI64SSA_QUERY_FSAMP_MIN	This returns the minimum sample rate, FSAMP, in samples per second.
AI64SSA_QUERY_INIT_MS	This returns the duration of a board initialization in milliseconds.
AI64SSA_QUERY_IRQ1_BUF_ERROR	This reports if the IRQ1 Buffer Error interrupt is supported.
AI64SSA_QUERY_LOW_LATENCY	This reports if the board supports the Low Latency feature.
AI64SSA_QUERY_MASTER_CLOCK	This returns the master clock frequency in hertz.
AI64SSA_QUERY_NRATE_MAX	This returns the maximum supported NRATE value.
AI64SSA_QUERY_NRATE_MIN	This returns the minimum supported NRATE value.
AI64SSA_QUERY_RATE_GEN_QTY	This returns the number of Rate Generators on the board.
AI64SSA_QUERY_REG_ACAR	This reports if the ACAR register is supported.

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

Value	Description
AI64SSA_IOCTL_QUERY_ERROR	Either there was a processing error or the query option is unrecognized.

4.7.37. AI64SSA_IOCTL_RAG_ENABLE

This service enables or disables the Rate-A Generator.

Usage

Argument	Description
request	AI64SSA_IOCTL_RAG_ENABLE
arg	s32*

Valid argument values are as follows.

Value	Description
-1	Retrieve the current setting.
AI64SSA_GEN_ENABLE_NO	This option disables the rate generator.
AI64SSA_GEN_ENABLE_YES	This option enables the rate generator.

4.7.38. AI64SSC_IOCTL_RAG_NRATE

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

Usage

Argument	Description
request	AI64SSC_IOCTL_RAG_NRATE
arg	s32*

Valid argument values are from a device specific minimum up to 0xFFFF, and -1. A value of -1 will return the current setting. The minimum value varies according to the board's master oscillator frequency and therefore is somewhere from a low of 150 to a maximum of 259. For the minimum NRATE for your board refer to the Query option AI64SSA_QUERY_NRATE_MIN (section 4.7.36, page 36).

4.7.39. AI64SSA_IOCTL_RBG_CLK_SRC

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

Usage

Argument	Description
request	AI64SSA_IOCTL_RBG_CLK_SRC
arg	s32*

Valid argument values are as follows.

Value	Description
-1	Retrieve the current setting.
AI64SSA_RBG_CLK_SRC_MASTER	This refers to the board's master clock.
AI64SSA_RBG_CLK_SRC_RAG	This refers to the Rate-A Generator output.

4.7.40. AI64SSA_IOCTL_RBG_ENABLE

This service enables or disables the Rate-B Generator.

Usage

Argument	Description
request	AI64SSA_IOCTL_RBG_ENABLE
arg	s32*

Valid argument values are as follows.

Value	Description
-1	Retrieve the current setting.
AI64SSA_GEN_ENABLE_NO	This option disables the rate generator.
AI64SSA_GEN_ENABLE_YES	This option enables the rate generator.

4.7.41. AI64SSC_IOCTL_RBG_NRATE

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

Usage

Argument	Description
request	AI64SSC_IOCTL_RBG_NRATE
arg	s32*

Valid argument values are from a device specific minimum up to 0xFFFF, and -1. A value of -1 will return the current setting. The minimum value varies according to the board's master oscillator frequency and therefore is somewhere from a low of 150 to a maximum of 259. For the minimum NRATE for your board refer to the Query option AI64SSA_QUERY_NRATE_MIN (section 4.7.36, page 36).

4.7.42. AI64SSA_IOCTL_REG_MOD

This service performs a read-modify-write of a 16AI64SSA register. This includes only the GSC firmware registers. The PCI and PLX Feature Set Registers are read-only. Refer to 16ai64ssa.h for the complete list of GSC firmware registers.

Usage

Argument	Description
request	AI64SSA_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.

4.7.43. AI64SSA_IOCTL_REG_READ

This service reads the value of a 16AI64SSA register. This includes the PCI registers, the PLX Feature Set Registers and the GSC firmware registers. Refer to 16ai64ssa.h, gsc_pci9056.h and gsc_pci9080.h for the complete list of accessible registers.

Usage

Argument	Description
request	AI64SSA_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.

4.7.44. AI64SSA_IOCTL_REG_WRITE

This service writes a value to a 16AI64SSA register. This includes only the GSC firmware registers. The PCI and PLX Feature Set Registers are read-only. Refer to `16ai64ssa.h` for a complete list of the GSC firmware registers.

Usage

Argument	Description
request	AI64SSA_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.

4.7.45. AI64SSA_IOCTL_RX_IO_ABORT

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

Usage

Argument	Description
request	AI64SSA_IOCTL_RX_IO_ABORT
arg	s32*

The results are reported as one of the following values.

Value	Description
AI64SSA_IO_ABORT_NO	A read() request was not aborted as none were ongoing.
AI64SSA_IO_ABORT_YES	An ongoing read() request was aborted.

4.7.46. AI64SSA_IOCTL_RX_IO_MODE

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

Usage

Argument	Description
request	AI64SSA_IOCTL_RX_IO_MODE
arg	s32*

Valid argument values are as follows.

Value	Description
-1	Retrieve the current setting.
GSC_IO_MODE_BMDMA	Use Block Mode DMA.
GSC_IO_MODE_DMDMA	Use Demand Mode DMA (transfer data as it becomes possible to do so).
GSC_IO_MODE_PIO	Use PIO mode, which is repetitive register access. This is the default.

4.7.47. AI64SSA_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

Argument	Description
request	AI64SSA_IOCTL_RX_IO_OVERFLOW
arg	s32*

Valid argument values are as follows.

Value	Description
-1	Retrieve the current setting.
AI64SSA_IO_OVERFLOW_CHECK	Perform the check. This is the default.
AI64SSA_IO_OVERFLOW_IGNORE	Do not perform the check.

4.7.48. AI64SSA_IOCTL_RX_IO_TIMEOUT

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

Usage

Argument	Description
request	AI64SSA_IOCTL_RX_IO_TIMEOUT
arg	s32*

Valid argument values are in the range from zero to 3600, -1, and AI64SSA_IO_TIMEOUT_INFINITE. A value of zero tells the driver not to sleep in order to wait for more data, and should only be used with PIO mode reads. A value of -1 is used to retrieve the current setting. If the option AI64SSA_IO_TIMEOUT_INFINITE is used, then the driver will wait indefinitely rather than timing out. The default is 10 seconds.

4.7.49. AI64SSA_IOCTL_RX_IO_UNDERFLOW

This service configures the read service to check for an input buffer underflow before performing the read operation. Extraneous sample data is returned 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

Argument	Description
request	AI64SSA_IOCTL_RX_IO_UNDERFLOW
arg	s32*

Valid argument values are as follows.

Value	Description
-1	Retrieve the current setting.
AI64SSA_IO_UNDERFLOW_CHECK	Perform the check. This is the default.
AI64SSA_IO_UNDERFLOW_IGNORE	Do not perform the check.

4.7.50. AI64SSA_IOCTL_SAMP_CLK_SRC

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

Usage

Argument	Description
request	AI64SSA_IOCTL_SAMP_CLK_SRC
arg	s32*

Valid argument values are as follows.

Value	Description
-1	Retrieve the current setting.
AI64SSA_SAMP_CLK_SRC_BCR	This refers to the Board Control Register's Input Sync bit.
AI64SSA_SAMP_CLK_SRC_EXT	This refers to the external clock input signal.
AI64SSA_SAMP_CLK_SRC_RAG	This refers to the Rate-A Generator output.
AI64SSA_SAMP_CLK_SRC_RBG	This refers to the Rate-B Generator output.

4.7.51. AI64SSA_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

Argument	Description
request	AI64SSA_IOCTL_SCAN_MARKER
arg	s32*

Valid argument values are as follows.

Value	Description
-1	Retrieve the current setting.
AI64SSA_SCAN_MARKER_DISABLE	Scan Markers are not inserted into the data stream.
AI64SSA_SCAN_MARKER_ENABLE	Scan Markers are inserted into the data stream.

4.7.52. AI64SSA_IOCTL_SCAN_MARKER_GET

This service retrieves the current Scan Marker value, which is inserted into the data stream when enabled. Refer to the board user manual for additional information.

Usage

Argument	Description
request	AI64SSA_IOCTL_SCAN_MARKER_GET
arg	u32*

Values returned are from zero to 0xFFFFFFFF.

4.7.53. AI64SSA_IOCTL_SCAN_MARKER_SET

This service sets the Scan Marker value, which is inserted into the data stream when enabled. Refer to the board user manual for additional information.

Usage

Argument	Description
request	AI64SSA_IOCTL_SCAN_MARKER_SET
arg	u32*

Valid argument values are from zero to 0xFFFFFFFF.

4.7.54. AI64SSA_IOCTL_WAIT_CANCEL

This service resumes all threads blocked via AI64SSA_IOCTL_WAIT_EVENT IOCTL calls (section 4.7.55, page 44), according to the provided criteria. When a blocked thread is waiting for any event specified in the structure, then the thread is resumed.

> **NOTE:** The driver itself makes use of the wait services for various internal operations. Driver initiated waits are unaffected by application cancel requests.

Usage

Argument	Description
request	AI64SSA_IOCTL_WAIT_CANCEL
arg	gsc_wait_t*

Definition

```
typedef struct
{
    u32  flags;
    u32  main;
    u32  gsc;
    u32  alt;
    u32  io;
    u32  timeout_ms;
    u32  count;
} gsc_wait_t;
```

Fields	Description
flags	This is unused by wait cancel operations.
main	This specifies the set of GSC_WAIT_MAIN_* events whose wait requests are to be cancelled. Refer to section 4.7.55.2 on page 45.
gsc	This specifies the set of AI64SSC_WAIT_GSC_* events whose wait requests are to be cancelled. Refer to section 4.7.55.3 on page 46.
alt	This is unused by the 16AI64SSC driver and should be zero.
io	This specifies the set of GSC_WAIT_IO_* events whose wait requests are to be cancelled. Refer to section 4.7.55.4 on page 46.
timeout_ms	This is unused by wait cancel operations.
count	Upon return this indicates the number of waits that were cancelled.

4.7.55. AI64SSA_IOCTL_WAIT_EVENT

This service blocks a thread until any one of a specified set of events occurs, or until a timeout lapses, whichever occurs first. The set of possible events to wait for are specified in the structure's main, gsc, alt and io fields. All field values must be valid and at least one event must be specified. If the thread is resumed because one of the referenced events has occurred, then the bit for the respective event is the only event bit that will be set. All other event bits and fields will be zero. (Multiple event bits will be set only if the events occur simultaneously.)

> **NOTE**: The service waits only for the first of the specified events, not for all specified events.

> **NOTE:** A wait timeout is reported via the gsc_wait_t structure's flags field having the GSC_WAIT_FLAG_TIMEOUT flag set, rather than via an ETIMEDOUT error.

Usage

Argument	Description
request	AI64SSA_IOCTL_WAIT_EVENT
arg	gsc_wait_t*

Definition

```
typedef struct
{
    u32  flags;
    u32  main;
    u32  gsc;
    u32  alt;
    u32  io;
    u32  timeout_ms;
    u32  count;
} gsc_wait_t;
```

Fields	Description
flags	This must initially be zero. Upon return this indicates the reason that the thread was resumed. Refer to section 4.7.55.1on page 45.
main	This specifies any number of GSC_WAIT_MAIN_* events that the thread is to wait for. Refer to section 4.7.55.2 on page 45.
gsc	This specifies any number of AI64SSC_WAIT_GSC_* events that the thread is to wait for. Refer to section 4.7.55.3 on page 46.
alt	This is unused by the 16AI64SSC driver and must be zero.
io	This specifies any number of GSC_WAIT_IO_* events that the thread is to wait for. Refer to section 4.7.55.4 on page 46.
timeout_ms	This specified the maximum amount of time, in milliseconds, that the thread is to wait for any of the referenced events. A value of zero means do not timeout at all. If non-zero, then upon return the value will be the approximate amount of time actually waited.
count	This is unused by wait event operations and must be zero.

4.7.55.1. gsc_wait_t.flags Options

Upon return from a wait request the wait structure's flags field will indicate the reason that the thread was resumed. Only one of the below options will be set.

Fields	Description
GSC_WAIT_FLAG_CANCEL	The wait request was cancelled.
GSC_WAIT_FLAG_DONE	One of the referenced events occurred.
GSC_WAIT_FLAG_TIMEOUT	The timeout period lapsed before a referenced event occurred.

4.7.55.2. 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 16AI64SSC and other General Standards products.

Fields	Description
GSC_WAIT_MAIN_DMA0	This refers to the DMA Done interrupt on DMA engine number zero.
GSC_WAIT_MAIN_DMA1	This refers to the DMA Done interrupt on DMA engine number one.
GSC_WAIT_MAIN_GSC	This refers to any of the Interrupt Control/Status Register interrupts.

`GSC_WAIT_MAIN_OTHER`	This generally refers to an interrupt generated by another device sharing the same interrupt as the 16AI64SSC.
`GSC_WAIT_MAIN_PCI`	This refers to any interrupt generated by the 16AI64SSC.
`GSC_WAIT_MAIN_SPURIOUS`	This refers to board interrupts which should never be generated.
`GSC_WAIT_MAIN_UNKNOWN`	This refers to board interrupts whose source could not be identified.

4.7.55.3. `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 selecting the desired interrupt options. Refer to `AI64SSA_IOCTL_IRQ0_SEL` (section 4.7.30, page 34) and `AI64SSA_IOCTL_IRQ1_SEL` (section 4.7.31, page 34).

Value	Description
`AI64SSA_WAIT_GSC_AUTOCAL_DONE`	This refers to the completion of an autocalibration cycle.
`AI64SSA_WAIT_GSC_BURST_DONE`	This refers to the completion of an input burst.
`AI64SSA_WAIT_GSC_BURST_START`	This refers to the beginning of an input burst.
`AI64SSA_WAIT_GSC_BUF_ERROR`	This refers to the occurrence of either an input buffer overflow or an input buffer underflow.
`AI64SSA_WAIT_GSC_IN_BUF_THR_H2L`	This refers to the input buffer threshold status being negated.
`AI64SSA_WAIT_GSC_IN_BUF_THR_L2H`	This refers to the input buffer threshold status being asserted.
`AI64SSA_WAIT_GSC_INIT_DONE`	This refers to the completion of an initialization cycle.
`AI64SSA_WAIT_GSC_SYNC_DONE`	This refers to the completion of a sync operation.
`AI64SSA_WAIT_GSC_SYNC_START`	This refers to the beginning of a sync operation.

4.7.55.4. `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
`AI64SSA_WAIT_IO_RX_ABORT`	This refers to read requests which have been aborted.
`AI64SSA_WAIT_IO_RX_DONE`	This refers to read requests which have been satisfied.
`AI64SSA_WAIT_IO_RX_ERROR`	This refers to read requests which end due to an error.
`AI64SSA_WAIT_IO_RX_TIMEOUT`	This refers to read requests which end due to the timeout period lapse.

4.7.56. AI64SSA_IOCTL_WAIT_STATUS

This service counts all threads blocked via the `AI64SSA_IOCTL_WAIT_EVENT` IOCTL service (section 4.7.55, page 44), according to the provided criteria. A match is made when a waiting thread's wait criteria matches any of the criteria specified in the structure passed to this service.

> **NOTE:** The driver itself makes use of the wait services for various internal operations. Driver initiated waits are ignored by application status requests.

Usage

Argument	Description
`request`	`AI64SSA_IOCTL_WAIT_STATUS`
`arg`	`gsc_wait_t*`

Definition

```
typedef struct
```

```
{
    u32  flags;
    u32  main;
    u32  gsc;
    u32  alt;
    u32  io;
    u32  timeout_ms;
    u32  count;
} gsc_wait_t;
```

Fields	Description
flags	This is unused by wait status operations.
main	This specifies the set of GSC_WAIT_MAIN_* events whose wait requests are to be counted. Refer to section 4.7.55.2 on page 45.
gsc	This specifies the set of AI64SSC_WAIT_GSC_* events whose wait requests are to be counted. Refer to section 4.7.55.3 on page 46.
alt	This is unused by the 16AI64SSC driver and should be zero.
io	This specifies the set of GSC_WAIT_IO_* events whose wait requests are to be counted. Refer to section 4.7.55.4 on page 46.
timeout_ms	This is unused by wait status operations.
count	Upon return this indicates the number of waits that met any of the specified criteria.

5. The Driver

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

5.1. Files

The device driver source files are summarized in the table below.

File	Description
`driver/*.c`	These are the driver source files.
`driver/*.h`	These are the driver header files.
`driver/start`	Shell script to install the driver executable and device nodes.
`driver/16ai64ssa.h`	A driver interface header file.
`driver/Makefile`	This is the driver make file.

5.2. 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 (…`/driver/`).

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

```
make clean
```

3. Build the driver by issuing the below command.

```
make
```

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

5.3. Startup

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

The startup script used in this procedure is designed to ensure that the driver module in the install directory is the module that is loaded. 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 ensure that the device nodes in use have the same major number as assigned dynamically to the driver by the kernel, and so that the number of device nodes correspond to the number of boards identified by the driver.

5.3.1. Manual Driver Startup Procedures

Start the driver manually by following the below listed steps.

NOTE: The following steps may require elevated privileges.

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

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

   ```
   ./start
   ```

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

   NOTE: The 16AI64SSA device node major number is assigned dynamically by the kernel. The minor numbers and the device node suffix numbers are index numbers beginning with zero, and increase by one for each additional board installed.

3. Verify that the device driver module has been loaded by issuing the below command and examining the output. The module name `16ai64ssa` should be included in the output.

   ```
   lsmod
   ```

4. Verify that the device nodes have been created by issuing the below command and examining the output. The output should include one node for each installed board.

   ```
   ls -l /dev/16ai64ssa.*
   ```

5.3.2. Automatic Driver Startup Procedures

Start the driver automatically with each system reboot by following the below listed steps.

1. Locate and edit the system startup script `rc.local`, which should be in the `/etc/rc.d/` directory. Modify the file by adding the below line so that it is executed with every reboot. The example is based on the driver being installed in `/usr/src/linux/drivers/`, though it may have been installed elsewhere.

   ```
   /usr/src/linux/drivers/16ai64ssa/driver/start
   ```

   NOTE: For `systemd` installations the file `rc.local` may be located under the `/etc/` directory rather than under `/etc/rc.d/`.

2. Load the driver and create the required device nodes by rebooting the system.

3. Verify that the driver is loaded and that the device nodes have been created. Do this by following the verification steps given in the manual startup procedures.

5.3.2.1. File `rc.local` Not Present

Some distributions may not install a default version of `rc.local`. Some may not even create the directory `/etc/rc.d/`. If the directory is not present, then it may be created. The directory must be created with the owner and group set to `root`. The directory permissions must be set to `rwxr-xr-x`. If the file `/etc/rc.d/rc.local` is not present, then it too may be created. The file must also be created with the owner and group set to `root`. Additionally, the file permissions must also be set to `rwxr-xr-x`. After the directory and file are created as described, reboot to verify boot time loading of the driver. Here is an example of a default version of `rc.local`.

```
#!/bin/bash

# Add you local content here.
```

5.3.2.2. Default rc.local File Permissions

The rc.local script may fail to run at boot time because some distributions install a default version of the file without execute permissions. Without execute permissions, boot time invocation of the script fails, which inhibits boot time loading of the driver. If this is the case, then change the file permissions to rwxr-xr-x. After the file permissions are adjusted as described, reboot to verify boot time loading of the driver.

5.3.2.3. systemd Installations

With the advent of the systemd startup implementation, rc.local may be accessed via a systemd startup service. The service name may be rc-local, rc-local.service or something similar. This service may or may not be enabled by default. If the service is disabled, then the script will not execute, which prevents boot time loading of the driver. The service can be enabled with the below command line. After the service is enabled, reboot to verify boot time loading of the driver.

```
systemctl enable rc-local
```

NOTE: For systemd installations the file rc.local may be located under the /etc/ directory rather than under /etc/rc.d/.

5.3.2.4. systemd and rc.local Timing

If the above steps have been performed but the driver still does not start then examine the dmesg output for driver messages. If the output shows that the driver starts and immediately stops, then the problem may be timing. That is, since systemd doesn't serialize startup initialization as done in the past, driver loading may fail if required services have not completed their own initialization. If this is the problem, then it may be corrected simply by inserting a delay in rc.local prior to it calling the driver's start script (i.e., sleep for one or more seconds).

5.3.2.5. SElinux Implications

If not disabled, then SElinux may prevent boot time loading of the driver. If this is the case, then it can be verified and corrected using SElinux related tools and utilities. First, install the necessary software using the below command. (As necessary, replace the yum command line with that which is available for your distribution.)

```
yum install setroubleshoot setools
```

Next, run the below command to determine if SElinux is preventing the driver from loading at boot time.

```
sealert –a /var/log/audit/audit.log
```

If SElinux is preventing the driver from loading, then the output from the above command should include a reference to the driver's start script, the insmod command that loads the driver or the name of the driver executable. If so, then the output should also indicate the commands necessary to resolve the issue. The following is an example of the instructions given when the culprit is insmod, which is the start script command that loads the driver. After running these commands reboot the system to verify boot time loading of the driver.

```
ausearch -c 'insmod' --raw | audit2allow -M my-insmod
semodule -X 300 -i my-insmod.pp
```

5.4. Verification

Follow the below steps to verify that the driver has been properly installed and started.

1. Verify that the file `/proc/16ai64ssa` 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/16ai64ssa
```

5.5. Version

The driver version number can be obtained in a variety of ways. It is reported by the driver both when the driver is loaded and when it is unloaded (depending on kernel configuration options, this may be visible only in places such as `/var/log/messages`). It is reported in the text file `/proc/16ai64ssa` while the driver is loaded and running.

5.6. Shutdown

Shutdown the driver following the below listed steps.

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

1. If the driver is currently loaded then issue the below command to unload the driver.

```
rmmod 16ai64ssa
```

2. Verify that the driver module has been unloaded by issuing the below command. The module name `16ai64ssa` should not be in the listed output.

```
lsmod
```

6. Document Source Code Examples

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

6.1. Files

The library files are summarized in the table below.

File	Description
`docsrc/*.c`	These are the C source files.
`docsrc/makefile`	This is the library make file.
`docsrc/makefile.dep`	This is an automatically generated make dependency file.
`include/16ai64ssa_dsl.h`	This is the primary utility header file.
`lib/16ai64ssa_dsl.a`	This is the statically linkable library file.

6.2. Build

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

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

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

   ```
   make clean
   ```

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

   ```
   make
   ```

6.3. Library Use

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

File	Location
`16ai64ssa_dsl.h`	…`/include/`
`16ai64ssa_dsl.a`	…`/lib/`

7. Utility Source Code

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

7.1. Files

The library files are summarized in the table below.

File	Description
`utils/util_*.c`	These are device specific utility source files.
`utils/gsc_*.c`	These are device and OS independent utility source files.
`utils/os_*.c`	These are OS specific utility source files.
`utils/makefile`	This is the library make file.
`utils/makefile.dep`	This is an automatically generated make dependency file.
`include/16ai64ssa_utils.h`	This is the primary utility header file.
`lib/16ai64ssa_utils.a`	This is the statically linkable library file.

7.2. Build

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

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

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

   ```
   make clean
   ```

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

   ```
   make
   ```

7.3. Library Use

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

File	Location
`16ai64ssa_utils.h`	`…/include/`
`16ai64ssa_utils.a`	`…/lib/`

8. Operating Information

This section explains some basic operational procedures for using the 16AI64SSA. This is in no way intended to be a comprehensive guide. This is simply to address a very few issues relating to their use.

8.1. Analog Input Configuration

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

Item	Name/File	Location
Function	ai64ssa_config_ai()	Source File
Source File	util_config_ai.c	…/utils/
Header File	16ai64ssa_utils.h	…/include/
Library File	16ai64ssa_utils.a	…/lib/

8.2. Auxiliary I/O Configuration

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

Item	Name/File	Location
Function	ai64ssa_config_aux()	Source File
Source File	util_config_aux.c	…/utils/
Header File	16ai64ssa_utils.h	…/include/
Library File	16ai64ssa_utils.a	…/lib/

8.3. I/O Modes

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

8.3.1. PIO

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

8.3.2. BMDMA - Block Mode DMA

For Block Mode DMA the driver initiates DMA transfers only after a sufficient volume of data has been received into the input buffer. In this mode the volume is sufficient when the input buffer content satisfies the request or when it meets or exceeds the threshold value. After that amount of data is in the input buffer the driver initiates a DMA then sleeps until the DMA Done interrupt is received. Using this DMA mode, a user request typically consists of numerous individual DMA transfers.

8.3.3. DMDMA - Demand Mode DMA

This DMA mode is similar to the block mode, except that the transfer is initiated immediately. Here however, the actual movement of data occurs as the data becomes available in the buffer instead of after it has been accumulated. Using this DMA mode, a user request typically consists of a single individual DMA transfer.

NOTE: On boards with a “PMC66-” or “PCIe-” model prefix DMDMA is performed based upon the availability of data as it appears in the input buffer. On boards with a “PMC-” model prefix DMDMA is performed based on the Input Buffer Threshold, which is effectively the same as using the DMA setting.

8.4. Debugging Aids

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

8.4.1. Device Identification

When communicating with technical support complete device identification is virtually always necessary. The *id* example application is provided for this specific purpose. This is a text only console application. The output can be piped to a file, which can then be emailed to GSC technical support when requested. Locate the application as follows.

Description	File	Location
Application	`id`	`…/id/`

8.4.2. Detailed Register Dump

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

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

Description	File/Name	Location
Function	`ai64ssa_reg_list()`	Source File
Source File	`util_reg.c`	`…/utils/`
Header File	`16ai64ssa_utils.h`	`…/include/`
Library File	`16ai64ssa_utils.a`	`…/lib/`

9. Sample Applications

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

9.1. fsamp - Sample Rate - …/fsamp/

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

9.2. id - Identify Board - …/id/

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

9.3. irq - Interrupt Test - …/irq/

This application performs complete testing to verify the operation of the board's firmware interrupts.

9.4. regs - Register Access - …/regs/

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

9.5. rxrate - Receive Rate - …/rxrate/

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

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

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

9.7. signals - Digital Signals - …/signals/

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

9.8. stream - Stream Rx Data to Disk - …/stream/

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

9.9. vrange - Voltage Range Test - …/vrange/

This application steps though all support input modes and voltage ranges, configuring the board accordingly, with a modest sample rate, then reads data from the board and reports the input data value ranges to the console. With suitable arguments and an input signal of approximately 3K Hz, the application will show an approximation of the input to the console.

9.10. wait - Wait Test - …/wait/

This application performs complete testing to verify the operation of the Wait Event options. This is similar to the `irq` application, but encompasses more interactions with the board.

Document History

Revision	Description
August 8, 2023	Updated to version 3.12.104.47.0. Updated the information for the open and close calls. Updated the kernel support table. Numerous, minor editorial changes. Updated the description of the Input Buffer Clear service. Updated the description of the Autocalibration service. Renamed AI64SSA_GSC_ACVR to AI64SSA_GSC_AVR. All Auto_Cal content renamed to Autocal. All Auto_Cal_Sts content renamed to Autocal_Status.
July 12, 2022	Updated to version 3.11.100.42.0. Expanded automatic startup information. Minor editorial corrections. Updated the kernel support table. Added section on environment variables.
October 22, 2020	Updated to version 3.10.91.35.0.
September 30, 2020	Updated to version 3.9.91.33.0. Updated the kernel support table. Corrected instances of incorrect model number. Minor editorial changes. Added a licensing subsection. Added WAIT_EVENT note. Expanded automatic startup information. Added incompatibility statements about using the differential input modes with the unipolar voltage ranges.
May 2, 2019	Updated to version 3.9.85.27.0. Minor editorial changes.
August 31, 2018	Updated to version 3.8.80.26.0. Updated the inside cover page. Updated the CPU and kernel support section. Various editorial changes. Updated Block Mode DMA macro and associated information. Added debugging aids.
February 27, 2018	Updated to version 3.7.74.21.0. Minor editorial changes. Updated the kernel support table.
December 28, 2017	Updated to version 3.7.73.20.0. Added `ai64ssa_init()` documentation.
September 11, 2017	Updated to version 3.7.72.20.0.
August 25, 2017	Updated to version 3.7.71.20.0. Restructured document.
November 8, 2016	Updated to version 3.6.68.18.0. Added notes regarding Demand Mode DMA. Added the `vrange` sample application. Added more arguments to the `savedata` application command line. Added support for infinite I/O timeouts. Updated the operating information section. Made various miscellaneous updates.
October 14, 2016	Updated to version 3.5.67.17.0.
September 28, 2016	Updated to version 3.4.67.17.0. Added cautionary notes about the IRQ Anomaly present in older firmware. Added the `irq` and `wait` sample applications. Added three new query options: AI64SSA_QUERY_CHAN_RANGE, AI64SSA_QUERY_REG_ACAR and AI64SSA_QUERY_IRQ1_BUF_ERROR.
September 22, 2016	Updated to version 3.3.67.17.0. Updated the command line arguments for the `fsamp`, `savedata` and `stream` sample applications. Updated the usage of the Wait Event `timeout_ms` field. Updated material on the open call. Added open access mode descriptions. Updated the kernel support table.
March 16, 2016	Updated to version 3.2.65.13.0.
January 21, 2016	Updated to version 3.2.64.12.0. Added the `-txv` option to the `stream` application.
November 11, 2015	Updated to version 3.1.64.12.0. Removed the `built` field from the `/proc` file. Updated the kernel support table. Removed the `sbtest` application section as it is not included in the release. Made version number changes.
September 11, 2015	Updated to version 3.0.60.8.0. Updated the Low Latency read service. Updated for the Low Latency Control Register, with services for the HOLD and RELEASE fields. Updated the device node name to include a period before the device index. Removed double underscore that prefaced various data types.
October 23, 2014	Updated to version 2.9.57.0.
February 26, 2014	Updated to version 2.9.52.0. Updated the kernel support data.
January 8, 2014	Updated to version 2.8.51.0. Updated the kernel support data.
November 13, 2013	Updated to version 2.8.49.0.
July 3, 2013	Updated to version 2.8.45.0.
May 11, 2013	Updated to version 2.8.42.0.
September 11, 2012	Updated to version 2.8.39.0.
July 18, 2012	Updated to version 2.7.39.0.

March 30, 2012	Updated to version 2.6.37.0.
March 29, 2012	Updated to version 2.5.37.0. Updated the kernel support data.
December 15, 2011	Updated to version 2.4.34.0. Corrected the `AI64SSA_IOCTL_CHAN_FIRST` and `AI64SSA_IOCTL_CHAN_LAST` service descriptions. Added the `gsc_utils.a` library.
December 8, 2011	Updated to version 2.3.33.0.
October 30, 2011	Updated to version 2.2.30.0.
July 27, 2011	Updated to version 2.1.28.0.
July 26, 2011	Updated to version 2.0.28.0. This is the initial release of the version 2.x.x.x driver.