# ADADIO

**8 A/D Channels, 4 D/A Channels, 16-bit**
**With 8-bit Discrete Digital I/O**

## PCI-ADADIO
## PMC-ADADIO
## PC104P-ADADIO

# Linux Device Driver
# User Manual

**Manual Revision: April 10, 2009**
**Driver Release 3.0.4.0**

**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

**General Standards Corporation** makes no warranty of any kind with regard to this material, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. Although extensive editing and reviews are performed before release to ECO control, **General Standards Corporation** assumes no responsibility for any errors that may exist in this document. No commitment is made to update or keep current the information contained in this document.

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

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

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



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

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

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

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

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

# Table of Contents

# 1. Introduction

This user manual applies to driver release 3.0.4.0.

## 1.1. Purpose

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

## 1.2. Acronyms

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

| Acronyms | Description |
|---|---|
| DMA | Direct Memory Access |
| GSC | General Standards Corporation |
| PCI | Peripheral Component Interconnect |
| PMC | PCI Mezzanine Card |

## 1.3. Definitions

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

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

## 1.4. Software Overview

The ADADIO driver software executes under control of the Linux operating system and runs in Kernel Mode as a Kernel Mode device driver. The ADADIO device driver is implemented as a standard dynamically loadable Linux device driver written in the C programming language. With the driver, user applications are able to open and close a device and, while open, perform read and I/O control operations.

## 1.5. Hardware Overview

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

## 1.6. Reference Material

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

- The applicable *ADADIO User Manual* from General Standards Corporation.

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

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

# 2. Installation

## 2.1. CPU and Kernel Support

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

| Kernel | Distribution | X86 | |
|---|---|---|---|
| | | 32-bit | 64-bit |
| 2.6.27 | SUSE 11.1 | Yes | Yes |
| 2.6.26 | Red Hat Fedora Core 9 (with updates as of 9/26/2008) | Yes | Yes |
| 2.6.25 | Red Hat Fedora Core 9 | Yes | Yes |
| 2.6.23 | Red Hat Fedora Core 8 | Yes | Yes |
| 2.6.21 | Red Hat Fedora Core 7 | Yes | Yes |
| 2.6.18 | Red Hat Fedora Core 6 | Yes | Yes |
| 2.6.16 | SUSE 10.1 | Yes | Yes |
| 2.6.15 | Red Hat Fedora Core 5 | Yes | Yes |
| 2.6.11 | Red Hat Fedora Core 4 | Yes | Yes |
| 2.6.9 | Red Hat Fedora Core 3 | Yes | Yes |
| 2.4.21 | Red Hat Enterprise Linux Workstation Release 3 | Yes | |
| 2.4.20 | Red Hat Linux 9 | Yes | |
| 2.4.18 | Red Hat Linux 7.3 | Yes | |
| 2.4.7 | Red Hat Linux 7.2 | Yes | |
| 2.2.14 | Red Hat Linux 6.2 | Yes | |

**NOTE:** The driver will have to be rebuilt 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 has not been tested for SMP operation.

### 2.1.1. 32-bit Support Under 64-bit Environments

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

## 2.2. The /proc File System

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

```
version: 3.0.4
built: Apr 10 2009, 15:21:14
32-bit support: yes (native)
boards: 1
models: ADADIO
```

| Entry | Description |
|---|---|
| version | This gives the driver version number in the form x.x.x. |
| built | This gives the driver build date and time as a string. It is given in the C form of printf("%s, %s", __DATE__, __TIME__). |
| 32-bit support: | This reports the driver's support for 32-bit applications. This will be either "yes" or "no" for 64-bit driver builds and "yes (native)" for 32-bit builds. |
| boards | This identifies the total number of boards the driver detected. |
| models | This gives a comma separated list of the basic model number for each board the driver detected. |

## 2.3. File List

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

| File | Description |
|---|---|
| adadio.linux.tar.gz | This archive contains the driver and all related sources. |
| adadio_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 Structure | Content |
|---|---|
| adadio | This is the source root directory. The user manual and overall make script are placed here. |
| adadio\aout | This directory contains the Analog Output application. Refer to section 5.1 on page 15. |
| adadio\docsrc | This directory contains the Document Source Code Examples. Refer to section 4 on page 14. |
| adadio\din | This directory contains the Digital Output application. Refer to section 5.2 on page 16. |
| adadio\dout | This directory contains the Digital Output application. Refer to section 5.3 on page 17. |
| adadio\driver | This directory contains the driver and its sources. Refer to section 3 on page 11. |
| adadio\id | This directory contains the Identification application. Refer to section 5.4 on page 18. |
| adadio\regs | This directory contains the Register Access application. Refer to section 5.5 on page 18. |
| adadio\rxrate | This directory contains the Receive Rate application. Refer to section 5.6 on page 19. |
| adadio\savedata | This directory contains the Save Data application. Refer to section 5.7 on page 20. |
| adadio\sbtest | This directory contains the Single Board Test application. Refer to section 5.8 on page 21. |
| adadio\testapp | This directory contains a sample application. Refer to section 5.9 on page 22. |

## 2.5. Installation

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

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

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

   ```
   tar –xzvf adadio.linux.tar.gz
   ```

## 2.6. Removal

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

1. Shutdown the driver as described in section 3.4 on page 12.

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 adadio.linux.tar.gz adadio
   ```

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

   ```
   rm –rf /dev/adadio*
   ```

5. If the automated startup procedure was adopted (see section 3.2.2 on page 12), then edit the system startup script `rc.local` and remove the line that invokes the ADADIO's `start` script. The file `rc.local` should be located in the `/etc/rc.d` directory.

## 2.7. Overall Make Script

An overall make script is included in the root installation directory. Executing this script will perform a make for all build targets included in the release, and it will also load the driver. The script is named `make_all`. Follow the below steps to perform an overall make and to load the driver.

1. Change to the device's root directory, which may be `/usr/src/linux/drivers/adadio`.

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

   ```
   ./make_all
   ```

# 3. The Driver

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

| File | Description |
|---|---|
| `driver/*.c` | The driver source files. |
| `driver/*.h` | The driver header files. |
| `driver/start` | Shell script to load the driver executable and create the device nodes. |
| `driver/adadio.h` | This is the main driver header file. This header should be included by ADADIO applications. |
| `driver/Makefile` | The driver make file. |

## 3.1. Build

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

Follow the below steps to build the driver.

1. Change to the directory where the driver and its sources are installed, which may be `/usr/src/linux/drivers/adadio/driver`.

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

   ```
   make clean
   ```

3. Build the driver by issuing the below command.

   ```
   make all
   ```

## 3.2. Startup

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

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

### 3.2.1. Manual Driver Startup Procedures

Start the driver manually by following the below listed steps.

1. Login as root user, as some of the steps require root privileges.

2. Change to the directory where the ADADIO driver sources are located, which may be `/usr/src/linux/drivers/adadio/driver.`

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

   ```
   ./start
   ```

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

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

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

   ```
   lsmod
   ```

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

   ```
   ls –l /dev/adadio*
   ```

### 3.2.2. Automatic Driver Startup Procedures

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

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

   ```
   /usr/src/linux/drivers/adadio/driver/start
   ```

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

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

### 3.2.3. Verification

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

1. Verify that the file `/proc/adadio` 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/adadio
   ```

## 3.3. 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/adadio` while the driver is loaded and running.

## 3.4. Shutdown

Shutdown the driver following the below listed steps.

1. Login as root user, as some of the steps require root privileges.

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

```
rmmod adadio
```

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

   ```
   lsmod
   ```

# 4. Document Source Code Examples

The archive file `adadio.linux.tar.gz` contains all of the source code examples included in this document. In addition, the code is built into a statically linkable library usable with ADADIO console applications. The library and sources are delivered undocumented and unsupported. The purpose of these files is to verify that the documentation samples compile and to provide a library of working sample code to assist in a user's learning curve and application development effort. These files are located in the `docsrc` subdirectory under the ADADIO root directory.

| File | Description |
|---|---|
| `docsrc/*.c` | These are the C source files. |
| `docsrc/adadio_dsl.h` | This is the library header file. |
| `docsrc/makefile` | This is the library make file. |
| `docsrc/makeile.dep` | This is an automatically generated make dependency file. |
| `docsrc/makeile.inc` | This is the library make include file. |

## 4.1. Build

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

1. Change to the directory where the documentation sources are installed, which may be `/usr/src/linux/drivers/adadio/docsrc`.

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

   ```
   make clean
   ```

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

   ```
   make all
   ```

## 4.2. Library Use

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

# 5. Sample Applications

The archive file `adadio.linux.tar.gz` contains all of the sample applications mentioned here, with all applicable sources.

## 5.1. Analog Output - aout

This sample application provides a command line driven Linux application that configures the analog output portion of a designated ADADIO board and outputs test patterns. The application is provided without documentation or support, but it can be used as the starting point for application development on top of the ADADIO Linux device driver. The application includes the below listed files.

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

### 5.1.1. Build

Follow the below steps to build the sample application.

1. Change to the directory where the sample application is installed, which may be `/usr/src/linux/drivers/adadio/aout`.

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

   ```
   make clean
   ```

3. Build the application by issuing the below command.

   ```
   make all
   ```

### 5.1.2. Execute

Follow the below steps to execute the sample application.

1. Change to the directory where the sample application is installed, which may be `/usr/src/linux/drivers/adadio/aout`.

2. Start the sample application by issuing the command given below. Once started the application will configure the board, generate a test pattern for each channel then output the pattern to the cable interface. The patterns generated are a saw tooth wave with a falling slope, a saw tooth wave with a rising slope, a square wave with a 50% duty cycle and a sine wave, respectively for output channels zero through three. The pattern length is reported to the screen in both samples and duration. The duration of each invocation depends on the arguments specified, but should be at least about 15 seconds. The command line arguments are described in the table below.

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

| Argument | Description |
|---|---|
| -c | Repeat the operation until an error is encountered. |
| -C | Repeat the operation, but continue even if errors are encountered. |
| -m# | When repeating the operation, stop after "#" minutes, where "#" is a decimal number. |
| -n# | When repeating the operation, stop after "#" interactions, where "#" is a decimal number. |
| -p# | This specifies the output period (or duration) in seconds, where "#" is a decimal number. |
| index | This is the index of the board to access. |

## 5.2. Digital Input - din

This sample application provides a command line driven Linux application that reads the cable's digital I/O signals and reports the values read to the screen. The application is provided without documentation or support, but it can be used as the starting point for application development on top of the ADADIO Linux device driver. The application includes the below listed files.

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

### 5.2.1. Build

Follow the below steps to build the sample application.

1. Change to the directory where the sample application is installed, which may be `/usr/src/linux/drivers/adadio/din`.

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

   ```
   make clean
   ```

3. Build the application by issuing the below command.

   ```
   make all
   ```

### 5.2.2. Execute

Follow the below steps to execute the sample application.

1. Change to the directory where the sample application is installed, which may be `/usr/src/linux/drivers/adadio/din`.

2. Start the sample application by issuing the command given below. Once started the application will configure the I/O signals as inputs, then repeatedly read the inputs and report the values to the screen. A single iteration should take about five seconds. The command line arguments are described in the table below.

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

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

## 5.3. Digital Output - dout

This sample application provides a command line driven Linux application that writes a pattern to the cable's digital I/O lines. The application is provided without documentation or support, but it can be used as the starting point for application development on top of the ADADIO Linux device driver. The application includes the below listed files.

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

### 5.3.1. Build

Follow the below steps to build the sample application.

1. Change to the directory where the sample application is installed, which may be `/usr/src/linux/drivers/adadio/dout`.

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

   ```
   make clean
   ```

3. Build the application by issuing the below command.

   ```
   make all
   ```

### 5.3.2. Execute

Follow the below steps to execute the sample application.

1. Change to the directory where the sample application is installed, which may be `/usr/src/linux/drivers/adadio/dout`.

2. Start the sample application by issuing the command given below. Once started the application will configure the I/O signals as outputs, then post a walking one pattern to the cable interface. A single iteration should take about 10 seconds. The command line arguments are described in the table below.

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

| Argument | Description |
|---|---|
| -c | Repeat the operation until an error is encountered. |
| -C | Repeat the operation, but continue even if errors are encountered. |
| -m# | When repeating the operation, stop after "#" minutes, where "#" is a decimal number. |

| | |
|---|---|
| -n# | When repeating the operation, stop after "#" interactions, where "#" is a decimal number. |
| index | This is the index of the board to access. |

## 5.4. Identify Board - id

This sample console application provides a command line driven Linux application that provides detailed board identification information. This can be used with tech support to help identify as much technical information about the board as possible from software. The application's sources are summarized in the below table.

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

### 5.4.1. Build

Follow the below steps to build the sample application.

1. Change to the directory where the sample application is installed, which may be /usr/src/linux/drivers/adadio/id.

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

   ```
   make clean
   ```

3. Build the application by issuing the below command.

   ```
   make all
   ```

### 5.4.2. Execute

Follow the below steps to execute the sample application.

1. Change to the directory where the sample application is installed, which may be /usr/src/linux/drivers/adadio/id.

2. Start the sample application by issuing the command given below. Once started the application will automatically output identification information. Execution should take about one second. The command line arguments are described in the table below.

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

| Argument | Description |
|---|---|
| index | This is the index of the board to access. |

## 5.5. Register Access - regs

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

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

### 5.5.1. Build

Follow the below steps to build the sample application.

1. Change to the directory where the sample application is installed, which may be `/usr/src/linux/drivers/adadio/regs`.

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

   ```
   make clean
   ```

3. Build the application by issuing the below command.

   ```
   make all
   ```

### 5.5.2. Execute

Follow the below steps to execute the sample application.

1. Change to the directory where the sample application is installed, which may be `/usr/src/linux/drivers/adadio/regs`.

2. Start the sample application by issuing the command given below. The command line argument is described in the table below.

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

| Argument | Description |
|---|---|
| index | This is the index of the board to access. |

## 5.6. Receive Rate – rxrate

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

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

### 5.6.1. Build

Follow the below steps to build the sample application.

1. Change to the directory where the sample application is installed, which may be `/usr/src/linux/drivers/adadio/rxrate`.

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

   ```
   make clean
   ```

3. Build the application by issuing the below command.

   ```
   make all
   ```

### 5.6.2. Execute

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

Follow the below steps to execute the sample application.

1. Change to the directory where the sample application is installed, which may be `/usr/src/linux/drivers/adadio/rxrate`.

2. Start the sample application by issuing the command given below. Once started the application will automatically performs an initialization of the board, then it will read and save the data. A single iteration should take less than 15 seconds, depending on the arguments given. The command line arguments are described in the table below.

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

| Argument | Description |
|---|---|
| `-c` | Repeat the operation until an error is encountered. |
| `-C` | Repeat the operation, but continue even if errors are encountered. |
| `-dma` | Perform the data transfer using DMA. |
| `-dmdma` | Perform the data transfer using Demand Mode DMA. |
| `-m#` | When repeating the operation, stop after "#" minutes, where "#" is a decimal number. |
| `-n#` | When repeating the operation, stop after "#" interactions, where "#" is a decimal number. |
| `-pio` | Perform the data transfer using PIO. |
| `-r#` | Retrieve "#" number of megabytes of data, where "#" is a decimal number. |
| `index` | This is the index of the board to access. |

## 5.7. Save Data - savedata

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

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

| | |
|---|---|
| utils/* | These are utility sources used by the application. |

### 5.7.1. Build

Follow the below steps to build the sample application.

1. Change to the directory where the sample application is installed, which may be `/usr/src/linux/drivers/adadio/savedata`.

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

   ```
   make clean
   ```

3. Build the application by issuing the below command.

   ```
   make all
   ```

### 5.7.2. Execute

Follow the below steps to execute the sample application.

1. Change to the directory where the sample application is installed, which may be `/usr/src/linux/drivers/adadio/savedata`.

2. Start the sample application by issuing the command given below. Once started the application will automatically performs an initialization of the board, then it will read and save the data. A single iteration should take only a few seconds. The command line arguments are described in the table below.

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

| Argument | Description |
|---|---|
| -c | Repeat the operation until an error is encountered. |
| -C | Repeat the operation, but continue even if errors are encountered. |
| -dma | Perform the data transfer using DMA. |
| -dmdma | Perform the data transfer using Demand Mode DMA. |
| -m# | When repeating the operation, stop after "#" minutes, where "#" is a decimal number. |
| -n# | When repeating the operation, stop after "#" interactions, where "#" is a decimal number. |
| -pio | Perform the data transfer using PIO. |
| index | This is the index of the board to access. |

## 5.8. Single Board Test - sbtest

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

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

### 5.8.1. Build

Follow the below steps to build the sample application.

1. Change to the directory where the sample application is installed, which may be `/usr/src/linux/drivers/adadio/sbtest`.

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

   ```
   make clean
   ```

3. Build the application by issuing the below command.

   ```
   make all
   ```

### 5.8.2. Execute

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

Follow the below steps to execute the sample application.

1. Change to the directory where the sample application is installed, which may be `/usr/src/linux/drivers/adadio/sbtest`.

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

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

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

## 5.9. Test Application - testapp

This sample application provides a command line driven Linux application that tests the functionality of the driver and a user specified ADADIO board. It can be used as the starting point for application development on top of the ADADIO Linux device driver. The application performs a series of control and I/O operations. The application includes the below listed files.

| File | Description |
|---|---|
| `testapp/testapp.c` | This is the application's source file. |
| `testapp/makefile` | This is the application make file. |
| `testapp/makefile.dep` | This is an automatically generated make dependency file. |
| `docsrc/*` | These are utility sources used by the application. |
| `utils/*` | These are utility sources used by the application. |

### 5.9.1. Build

Follow the below steps to build the sample application.

1. Change to the directory where the sample application is installed, which may be `/usr/src/linux/drivers/adadio/testapp`.

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

   ```
   make clean
   ```

3. Build the application by issuing the below command.

   ```
   make all
   ```

### 5.9.2. Execute

Follow the below steps to execute the sample application.

1. Change to the directory where the sample application is installed, which may be `/usr/src/linux/drivers/adadio/testapp`.

2. Start the sample application by issuing the command given below. The command line arguments are given in the table below. The application will perform a series of setup operations and then begin reading data. The application will terminate after any key is pressed.

   ```
   ./testapp <index>
   ```

| Argument | Description |
|---|---|
| `index` | This is the index of the board to access. |

# 6. Driver Interface

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

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

## 6.1. Macros

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

### 6.1.1. IOCTL Services

The IOCTL macros are documented in section 6.4 beginning on page 28.

### 6.1.2. Registers

The following gives the complete set of ADADIO registers.

#### 6.1.2.1. GSC Registers

The following table gives the complete set of GSC specific ADADIO registers. For detailed definitions of these registers refer to the *ADADIO User Manual*.

| Macro | Description |
|---|---|
| ADADIO_GSC_AIDR | Analog Input Data Register |
| ADADIO_GSC_AOC0R | Analog Output Channel 0 Register |
| ADADIO_GSC_AOC1R | Analog Output Channel 1 Register |
| ADADIO_GSC_AOC2R | Analog Output Channel 2 Register |
| ADADIO_GSC_AOC3R | Analog Output Channel 3 Register |
| ADADIO_GSC_BCR | Board Control Register |
| ADADIO_GSC_BRR | Board Revision Register * |
| ADADIO_GSC_DIOPR | Digital I/O Port Register |
| ADADIO_GSC_SRR | Sample Rate Register |

* The first time this register is read after a fresh load of the driver may take several seconds as access to this register requires an Auto-Calibration cycle and an initialization.

#### 6.1.2.2. PCI Configuration Registers

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

#### 6.1.2.3. PLX PCI9080 Feature Set Registers

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

## 6.2. Data Types

The data types used by the driver interface are defined along with the IOCTL services with which they are used. For additional information on the supported IOCTL services refer to section 6.4 beginning on page 28.

## 6.3. Functions

This driver interface includes the following functions.

### 6.3.1. close()

This function is the entry point to close a connection to an open ADADIO board. This function should only be called after a successful open of the respective device.

Prototype

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

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

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

Example

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

#include "adadio_dsl.h"

int adadio_dsl_close(int fd)
{
    int status;

    status  = close(fd);

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

    return(status);
}
```

### 6.3.2. ioctl()

This function is the entry point to performing setup and control operations on an ADADIO board. This function should only be called after a successful open of the respective device. The specific operation performed varies according to the request argument. The request argument also governs the use and interpretation of any additional arguments. The set of supported IOCTL services is defined in section 6.4 beginning on page 28.

Prototype

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

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

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

Example

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

#include "adadio_dsl.h"

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

    status  = ioctl(fd, request, arg);

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

    return(status);
}
```

### 6.3.3. open()

This function is the entry point to open a connection to an ADADIO board. The pathname to an ADADIO board is `/dev/adadio`*n*, where the trailing "n" is the zero based index of the board to access.

Prototype

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

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

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

| Return Value | Description |
|---|---|
| -1 | An error occurred. Consult errno. |
| else | A valid file descriptor. |

Example

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

#include "adadio_dsl.h"

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

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

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

    return(fd);
}
```

### 6.3.4. read()

This function is the entry point to reading data from an open ADADIO. This function should only be called after a successful open of the respective device. The function reads up to count bytes from the board. The return value is the number of bytes actually read. Refer to section 7.1 on page 40 for data transfer option information. (Refer to the IOCTL service descriptions as a number of them affect I/O operations. See section 6.4 beginning on page 28.)

Prototype

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

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

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

Example

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

#include "adadio_dsl.h"

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

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

    if (status >= 0)
        status  /= 4;
    else
        printf("read() failure, errno = %d\n", errno);

    return(status);
}
```

#### 6.3.5. write()

This function is not supported. To output analog data an application should use the `ADADIO_IOCTL_AOUT_CH_`*`X`*`_WRITE` IOCTL services (section 6.4.8, page 31).

## 6.4. IOCTL Services

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

#### 6.4.1. ADADIO_IOCTL_AIN_BUF_CLEAR

This service clears the data from the input buffer.

Usage

| **`ioctl()` Argument** | **Description** |
|---|---|
| `request` | `ADADIO_IOCTL_AIN_BUF_CLEAR` |
| `arg` | Not used. |

#### 6.4.2. ADADIO_IOCTL_AIN_BUF_SIZE

This service sets the size of the board's virtual input buffer. The physical buffer size is 32K samples deep.

**NOTE:** The buffer fill level status flags refer to the virtual buffer size, not the physical buffer size.

**NOTE:** Input sample collection is halted while the virtual buffer is full.

Usage

| **ioctl() Argument** | **Description** |
|---|---|
| request | ADADIO_IOCTL_AIN_BUF_SIZE |
| arg | __s32* |

Valid argument values are as follows.

| **Value** | **Description** |
|---|---|
| -1 | This requests the current setting. |
| ADADIO_AIN_BUF_SIZE_1 | Set the Virtual Buffer size to one sample. |
| ADADIO_AIN_BUF_SIZE_2 | Set the Virtual Buffer size to two samples. |
| ADADIO_AIN_BUF_SIZE_4 | Set the Virtual Buffer size to four samples. |
| ADADIO_AIN_BUF_SIZE_8 | Set the Virtual Buffer size to eight samples. |
| ADADIO_AIN_BUF_SIZE_16 | Set the Virtual Buffer size to 16 samples. |
| ADADIO_AIN_BUF_SIZE_32 | Set the Virtual Buffer size to 32 samples. |
| ADADIO_AIN_BUF_SIZE_64 | Set the Virtual Buffer size to 64 samples. |
| ADADIO_AIN_BUF_SIZE_128 | Set the Virtual Buffer size to 128 samples. |
| ADADIO_AIN_BUF_SIZE_256 | Set the Virtual Buffer size to 256 samples. |
| ADADIO_AIN_BUF_SIZE_512 | Set the Virtual Buffer size to 512 samples. |
| ADADIO_AIN_BUF_SIZE_1024 | Set the Virtual Buffer size to 1,024 samples. |
| ADADIO_AIN_BUF_SIZE_2048 | Set the Virtual Buffer size to 2,048 samples. |
| ADADIO_AIN_BUF_SIZE_4096 | Set the Virtual Buffer size to 4,096 samples. |
| ADADIO_AIN_BUF_SIZE_8192 | Set the Virtual Buffer size to 8,192 samples. |
| ADADIO_AIN_BUF_SIZE_16384 | Set the Virtual Buffer size to 16,384 samples. |
| ADADIO_AIN_BUF_SIZE_32768 | Set the Virtual Buffer size to 32,768 samples, which is the buffer's physical size. |

### 6.4.3. ADADIO_IOCTL_AIN_BUF_STS

This service retrieves the fill level status of the virtual input buffer.

Usage

| **ioctl() Argument** | **Description** |
|---|---|
| request | ADADIO_IOCTL_AIN_BUF_STS |
| arg | __s32* |

Valid argument values returned are as follows.

| **Value** | **Description** |
|---|---|
| ADADIO_AIN_BUF_STS_EMPTY | The virtual buffer is empty. |
| ADADIO_AIN_BUF_STS_ALMOST_EMPTY | The buffer is less than half full, but it is not empty. |
| ADADIO_AIN_BUF_STS_HALF_FULL | The buffer is at least half full, but it is not full. |
| ADADIO_AIN_BUF_STS_FULL | The virtual buffer is full. |

### 6.4.4. ADADIO_IOCTL_AIN_CHAN_LAST

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

Usage

| ioctl() Argument | Description |
|---|---|
| request | ADADIO_IOCTL_AIN_CHAN_LAST |
| arg | __s32* |

Valid argument values are as follows.

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

### 6.4.5. ADADIO_IOCTL_AIN_MODE

This service configures the analog input mode.

Usage

| ioctl() Argument | Description |
|---|---|
| request | ADADIO_IOCTL_AIN_MODE |
| arg | __s32* |

Valid argument values are as follows.

| Value | Description |
|---|---|
| -1 | Passing in this value requests the current setting. |
| ADADIO_AIN_MODE_DIFF_BURST | This refers to Differential, burst input operation. |
| ADADIO_AIN_MODE_DIFF_CONT | This refers to Differential, continuous input operation. |
| ADADIO_AIN_MODE_LB_TEST | This refers to connection of a single output channel to all input channels. |
| ADADIO_AIN_MODE_SE_BURST | This refers to Single Ended, burst input operation. |
| ADADIO_AIN_MODE_SE_CONT | This refers to Single Ended, continuous input operation. |
| ADADIO_AIN_MODE_VREF_TEST | This option connects the inputs to the +VREF reference voltage. |
| ADADIO_AIN_MODE_ZERO_TEST | This option connects the inputs to the zero reference voltage. |

### 6.4.6. ADADIO_IOCTL_AIN_NRATE

This service sets the NRATE divider value for input sample rate generation.

Usage

| ioctl() Argument | Description |
|---|---|
| request | ADADIO_IOCTL_AIN_NRATE |
| arg | __s32* |

Valid argument values are as follows.

| Value | Description |
|---|---|
| -1 | Passing in this value requests the current setting. |
| 100 – 0xFFFF | This is the range for 200K S/S boards |
| 200 – 0xFFFF | This is the range for 100K S/S boards |

### 6.4.7. ADADIO_IOCTL_AIN_TRIGGER

This service initiates an input strobe operation.

Usage

| ioctl() Argument | Description |
|---|---|
| request | ADADIO_IOCTL_AIN_TRIGGER |
| arg | Not used. |

### 6.4.8. ADADIO_IOCTL_AOUT_CH_*X*_WRITE

This refers to the below listed services.

| Service | Description |
|---|---|
| ADADIO_IOCTL_AOUT_CH_0_WRITE | Write to Output Channel 0. |
| ADADIO_IOCTL_AOUT_CH_1_WRITE | Write to Output Channel 1. |
| ADADIO_IOCTL_AOUT_CH_2_WRITE | Write to Output Channel 2. |
| ADADIO_IOCTL_AOUT_CH_3_WRITE | Write to Output Channel 3. |

These services write a value to their respective analog output channels.

Usage

| ioctl() Argument | Description |
|---|---|
| request | ADADIO_IOCTL_AOUT_CH_*X*_WRITE |
| arg | __s32* |

Valid argument values are as follows.

| Value | Description |
|---|---|
| -1 | Passing in this value requests the current setting. |
| 0x0000 – 0xFFFF | This is the value to be applied to the output channel. |

### 6.4.9. ADADIO_IOCTL_AOUT_ENABLE

This service enables or disables the analog outputs.

Usage

| Ioctl() Argument | Description |
|---|---|
| request | ADADIO_IOCTL_AOUT_ENABLE |
| arg | __s32* |

Valid argument values are as follows.

| Value | Description |
|---|---|
| -1 | Passing in this value requests the current setting. |
| ADADIO_AOUT_ENABLE_NO | Disable the outputs. |

| | |
|---|---|
| ADADIO_AOUT_ENABLE_YES | Enable the outputs |

### 6.4.10. ADADIO_IOCTL_AOUT_STROBE

This service initiates an output strobe operation.

Usage

| **ioctl() Argument** | **Description** |
|---|---|
| request | ADADIO_IOCTL_AOUT_STROBE |
| arg | Not used. |

### 6.4.11. ADADIO_IOCTL_AOUT_STROBE_ENABLE

This service enables or disables output strobe operation.

Usage

| **Ioctl() Argument** | **Description** |
|---|---|
| request | ADADIO_IOCTL_AOUT_STROBE_ENABLE |
| arg | __s32* |

Valid argument values are as follows.

| **Value** | **Description** |
|---|---|
| -1 | Passing in this value requests the current setting. |
| ADADIO_AOUT_STROBE_ENABLE_NO | Disable output strobe operation. Values are posted to the output immediately. |
| ADADIO_AOUT_STROBE_ENABLE_YES | Enable output strobe operation. Values are posted to the output only in response to an output strobe. |

### 6.4.12. ADADIO_IOCTL_AUTO_CALIBRATE

This service initiates an Auto-Calibration cycle. The service returns when the operation completes, which is on the order of about 10 seconds.

Usage

| **ioctl() Argument** | **Description** |
|---|---|
| request | ADADIO_IOCTL_AUTO_CALIBRATE |
| arg | Not used. |

### 6.4.13. ADADIO_IOCTL_DATA_FORMAT

This service sets the analog input and output data encoding format.

Usage

| **ioctl() Argument** | **Description** |
|---|---|
| request | ADADIO_IOCTL_DATA_FORMAT |
| arg | __s32* |

Valid argument values returned are as follows.

| Value | Description |
|---|---|
| -1 | This requests the current setting. |
| ADADIO_DATA_FORMAT_2S_COMP | This refers to the Twos Compliment encoding format. |
| ADADIO_DATA_FORMAT_OFF_BIN | This refers to the Offset Binary encoding format. |

#### 6.4.14. ADADIO_IOCTL_DIO_PIN_READ

This service reads the dedicated digital input pin.

Usage

| ioctl() Argument | Description |
|---|---|
| request | ADADIO_IOCTL_DIO_PIN_READ |
| arg | __s32* |

Valid argument values returned are as follows.

| Value | Description |
|---|---|
| ADADIO_DIO_PIN_CLEAR | The dedicated input is low. |
| ADADIO_DIO_PIN_SET | The dedicated input is high. |

#### 6.4.15. ADADIO_IOCTL_DIO_PIN_WRITE

This service sets the output level for the dedicated digital output pin.

Usage

| ioctl() Argument | Description |
|---|---|
| request | ADADIO_IOCTL_DIO_PIN_WRITE |
| arg | __s32* |

Valid argument values are as follows.

| Value | Description |
|---|---|
| -1 | This requests the current setting. |
| ADADIO_DIO_PIN_CLEAR | Set the output to a low level. |
| ADADIO_DIO_PIN_SET | Set the output to a high level. |

#### 6.4.16. ADADIO_IOCTL_DIO_PORT_DIR

This service sets the direction of the 8-bit digital I/O port port.

Usage

| ioctl() Argument | Description |
|---|---|
| request | ADADIO_IOCTL_DIO_PORT_DIR |
| arg | __s32* |

Valid argument values are as follows.

| Value | Description |
|---|---|
| -1 | This requests the current setting. |
| ADADIO_DIO_PORT_DIR_INPUT | This selects the input direction. |
| ADADIO_DIO_PORT_DIR_OUTPUT | This selects the output direction. |

### 6.4.17. ADADIO_IOCTL_DIO_PORT_READ

This service reads the value at the digital I/O port. The value read is input data only if the port is configured as an input. If the port is configured to output data, then this service retrieves the current output value.

Usage

| **ioctl() Argument** | **Description** |
|---|---|
| request | ADADIO_IOCTL_DIO_PORT_READ |
| arg | __s32* |

Valid argument values returned are as follows.

| **Value** | **Description** |
|---|---|
| 0x00 – 0xFF | This is an 8-bit port. |

### 6.4.18. ADADIO_IOCTL_DIO_PORT_WRITE

This service sets the output value for the digital I/O port. The value provided appears at the digital I/O port only if the port is configured as an output.

Usage

| **Ioctl() Argument** | **Description** |
|---|---|
| request | ADADIO_IOCTL_DIO_PORT_WRITE |
| arg | __s32* |

Valid argument values are as follows.

| **Value** | **Description** |
|---|---|
| -1 | Passing in this value returns the last output value, if the port is configured as an output. If the port is configured as an input, then this will retrieve the current input. |
| 0x00 – 0xFF | This is an 8-bit port. |

### 6.4.19. ADADIO_IOCTL_INITIALIZE

This service performs board initialization. The service returns when the operation completes, which is on the order of a few milliseconds.

Usage

| **ioctl() Argument** | **Description** |
|---|---|
| request | ADADIO_IOCTL_INITIALIZE |
| arg | Not used. |

### 6.4.20. ADADIO_IOCTL_IRQ_ENABLE

This service enables or disables firmware interrupts, which is the master for the interrupt specified in the Board Control Register.

Usage

| **ioctl() Argument** | **Description** |
|---|---|
| request | ADADIO_IOCTL_IRQ_ENABLE |

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

Valid argument values are as follows.

| Value | Description |
|---|---|
| -1 | Retrieve the current status. |
| ADADIO_IRQ_ENABLE_NO | This option prevents the firmware interrupt from being passed to the processor. The interrupt is still operational, but it will not result in the processor being interrupted. |
| ADADIO_IRQ_ENABLE_YES | This option permits the firmware interrupt to be passed to the processor. |

### 6.4.21. ADADIO_IOCTL_IRQ_SEL

This service configures the source selection for firmware interrupts.

Usage

| **ioctl() Argument** | **Description** |
|---|---|
| request | ADADIO_IOCTL_IRQ_SEL |
| arg | __s32* |

Valid argument values are as follows.

| Value | Description |
|---|---|
| -1 | Retrieve the current setting. |
| ADADIO_IRQ_SEL_AUTO_CAL_DONE | This refers to the completion of an Auto-Calibration cycle. |
| ADADIO_IRQ_SEL_AIN_BUF_EMPTY | This refers to the Input Buffer becoming empty. |
| ADADIO_IRQ_SEL_AIN_BUF_FULL | This refers to the Input Buffer becoming full. |
| ADADIO_IRQ_SEL_AIN_BUF_HALF_FULL | This refers to the Input Buffer becoming half full. |
| ADADIO_IRQ_SEL_AIN_BURST_DONE | This refers to the completion of an input burst. |
| ADADIO_IRQ_SEL_AOUT_STROBE_DONE | This refers to the completion of an output strobe. |
| ADADIO_IRQ_SEL_INIT_DONE | This refers to the completion of an initialization cycle. |

### 6.4.22. ADADIO_IOCTL_IRQ_STATUS

This service retrieves the status of the firmware interrupt.

Usage

| **ioctl() Argument** | **Description** |
|---|---|
| request | ADADIO_IOCTL_IRQ_STATUS |
| arg | __s32* |

Valid argument values are as follows.

| Value | Description |
|---|---|
| -1 | Retrieve the current state. |
| ADADIO_IRQ_STATUS_CLEAR | This option will clear the interrupt status if it is set. |
| ADADIO_IRQ_STATUS_IGNORE | This option takes no action regarding the current state. |

When retrieving the current state the below values are returned.

| Value | Description |
|---|---|
| ADADIO_IRQ_STATUS_ACTIVE | An interrupt has been generated. |
| ADADIO_IRQ_STATUS_IDLE | An interrupt has not been generated. |

### 6.4.23. ADADIO_IOCTL_LOOPBACK_CHANNEL

This service selects the output channel to use for the Loopback input mode selection.

Usage

| ioctl() Argument | Description |
|---|---|
| request | ADADIO_IOCTL_LOOPBACK_CHANNEL |
| arg | __s32* |

Valid argument values are as follows.

| Value | Description |
|---|---|
| -1 | This requests the current setting. |
| 0 | Select channel 0 output as the input source. |
| 1 | Select channel 1 output as the input source. |
| 2 | Select channel 2 output as the input source. |
| 3 | Select channel 3 output as the input source. |

### 6.4.24. ADADIO_IOCTL_QUERY

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

Usage

| ioctl() Argument | Description |
|---|---|
| request | ADADIO_IOCTL_QUERY |
| arg | __s32* |

Valid argument values are as follows.

| Value | Description |
|---|---|
| ADADIO_QUERY_AUTO_CAL_MS | This returns the maximum duration of the Auto Calibration cycle in milliseconds. |
| ADADIO_QUERY_COUNT | This returns the number of query options supported by the IOCTL service. |
| ADADIO_QUERY_DEVICE_TYPE | This returns the identifier value for the board's type. This should be GSC_DEV_TYPE_ADADIO. |
| ADADIO_QUERY_DMDMA | Does the board support Demand Mode DMA? (0 = no, 1 = yes) |
| ADADIO_QUERY_FIFO_SIZE | This returns the size of the input buffer in 32-bit A/D values. |
| ADADIO_QUERY_FSAMP_MAX | This gives the maximum Fsamp value in S/S. |
| ADADIO_QUERY_FSAMP_MIN | This gives the minimum Fsamp value in S/S. |
| ADADIO_QUERY_INIT_MS | This returns the duration of a board initialization in milliseconds. |
| ADADIO_QUERY_MASTER_CLOCK | This returns the master clock frequency in hertz. |
| ADADIO_QUERY_NRATE_MASK | This returns the mask for the board's NRATE fields. |
| ADADIO_QUERY_NRATE_MAX | This returns the maximum supported NRATE value. |
| ADADIO_QUERY_NRATE_MIN | This returns the minimum supported NRATE value. |

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

| Value | Description |
|---|---|
| ADADIO_IOCTL_QUERY_ERROR | Either there was a processing error or the query option is unrecognized. |

### 6.4.25. ADADIO_IOCTL_REG_MOD

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

Usage

| ioctl() Argument | Description |
|---|---|
| request | ADADIO_IOCTL_REG_MOD |
| arg | gsc_reg_t* |

Definition

```
typedef struct
{
    __u32   reg;    // range: any valid register definition
    __u32   value;  // range: 0x0-0xFFFFFFFF
    __u32   mask;   // range: 0x0-0xFFFFFFFF
} gsc_reg_t;
```

| Fields | Description |
|---|---|
| reg | This is the register to access. Refer to section 6.1.2 on page 24 for additional information. |
| value | This is the value to write to the specified register. Only the bits set in the map are applied. |
| mask | This is a map of the bits to modify. If a bit is set, then the corresponding register bit is set according the content of the value field. If a bit here is zero, then that register bit is unmodified. |

### 6.4.26. ADADIO_IOCTL_REG_READ

This service reads the value of an ADADIO register. This includes the PCI registers, the PLX Feature Set registers and the firmware registers.

Usage

| ioctl() Argument | Description |
|---|---|
| request | ADADIO_IOCTL_REG_READ |
| arg | gsc_reg_t* |

Definition

```
typedef struct
{
    __u32   reg;    // range: any valid register definition
    __u32   value;  // range: 0x0-0xFFFFFFFF
    __u32   mask;   // range: 0x0-0xFFFFFFFF
} gsc_reg_t;
```

| Fields | Description |
|---|---|
| reg | This is the register to read from. Refer to section 6.1.2 on page 24 for additional information. |
| value | This is the value read from the specified register. |

| | |
|---|---|
| mask | This is ignored for read requests. |

### 6.4.27. ADADIO_IOCTL_REG_WRITE

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

Usage

| ioctl() Argument | Description |
|---|---|
| request | ADADIO_IOCTL_REG_WRITE |
| arg | gsc_reg_t* |

Definition

```
typedef struct
{
    __u32   reg;    // range: any valid register definition
    __u32   value;  // range: 0x0-0xFFFFFFFF
    __u32   mask;   // range: 0x0-0xFFFFFFFF
} gsc_reg_t;
```

| Fields | Description |
|---|---|
| reg | This is the register to write to. Refer to section 6.1.2 on page 24 for additional information. |
| value | This is the value to write to the specified register. |
| mask | This is ignored for write requests. |

### 6.4.28. ADADIO_IOCTL_RX_IO_MODE

This service selects the data transfer mode for I/O operations. Refer to the read() service for additional information (section 6.3.4 on page 27).

Usage

| ioctl() Argument | Description |
|---|---|
| request | ADADIO_IOCTL_RX_IO_MODE |
| arg | __s32* |

Valid argument values are as follows.

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

**NOTE:** Demand Mode DMA is not available on boards with older firmware. Refer to the board hardware manual for details, or to the ADADIO_QUERY_DMDMA query option (section 6.4.24, page 36).

### 6.4.29. ADADIO_IOCTL_RX_IO_TIMEOUT

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

Usage

| **ioctl() Argument** | **Description** |
|---|---|
| request | ADADIO_IOCTL_RX_IO_TIMEOUT |
| arg | __s32* |

Valid argument values are from zero to 3600. Passing in a value of -1 retrieves the current setting.

# 7. Operation

This section explains some operational procedures using the driver. This is in no way intended to be a comprehensive guide on using the ADADIO. This is simply to address a few issues relating to using the ADADIO.

## 7.1. Data Transfer Options

### 7.1.1. Programmed I/O

This is referred to as PIO. In this mode data is transferred using repetitive register accesses. This is most applicable for low throughput requirements or for small transfer requests. The driver will read data from the input buffer register until either the buffer is empty, or the I/O timeout expires, whichever occurs first. This is generally the least efficient mode, but for very small transfers it is more efficient than DMA.

### 7.1.2. Non-Demand Mode DMA

This mode is intended for data transfers that do not exceed the size of the ADADIO input buffer. Here, the board's DMA engine is used to perform a hardware controlled transfer which does not require processer intervention to move the data. In this mode the DMA transfer is initiated only when the input buffer contains sufficient data to fulfill the request. This is a very efficient I/O method. However, for small requests PIO is more efficient.

### 7.1.3. Demand Mode DMA

This DMA transfer mode is similar to the Non-Demand mode, except that a transfer for the entire amount of data is initiated immediately and is not limited to the size of the virtual FIFO. Here however, the actual movement of data occurs as the data becomes available in the input buffer. This is the most efficient method supported. However, for small requests PIO is more efficient.

## Document History

| Revision | Description |
|---|---|
| April 10, 2009 | Updated to release 3.0.4.0. Extensive interface changes. Added sample applications. |
| September 2, 2008 | Updated to release 2.1.0. Numerous modifications. |
| November 2, 2004 | Updated to release 2.0.0. |
| February 11, 2003 | Ported the driver to the 2.4 kernel. |
| February 10, 2003 | The test application now uses `ADADIODocSrcLib`. Some code examples were updated. |
| February 7, 2003 | Added notes about `mmap()` of GSC registers when they aren't on a page boundary. The documentation sources are now included as a library. |
| June 13, 2002 | Initial release. |