SIOB4/8-SYNC

Four Channel High Speed Serial I/O

All SIO4 and SIO8 Models
All Form Factors
All Standard SYNC Versions

Linux Device Driver
User Manual

Manual Revision: June 13, 2023
Driver Release Version 1.59.104.47.0

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

1. Introduction

This user manual applies to the SYNC specific support provided by the driver. This user manual is intended for those SIO4 and SIO8 models that support the –SYNC firmware. This includes those boards with the –SYNC in the model number as well as those model boards that support software configurable firmware selection (see explanation, section 6.4.3, page 44).

> **NOTE:** The device models listed on the front cover are those that are specifically supported by this release of the driver. Other models may be supported, though the level of support may vary. The driver may work with other SIO4 models, but performance may be degraded due to device feature and implementation differences.

1.1. Purpose

The purpose of this document is to describe the interface to the SIO4 Linux Device Driver and SYNC library, as it relates to SYNC versions of the SIO4. This software provides the interface between "Application Software" and the SIO4 board. The interface to this board is at the device level.

1.2. Acronyms

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

Acronyms	Description
BMDMA	Block Mode DMA
DMA	Direct Memory Access
DMDMA	Demand Mode DMA
GSC	General Standards Corporation
LSBF	Least Significant Bit First
MSBF	Most Significant Bit First
PCI	Peripheral Component Interconnect
PMC	PCI Mezzanine Card

1.3. Definitions

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

Term	Definition
Application	Application means the user mode process, which runs in user space with user mode privileges.
Driver	Driver means the kernel mode device driver, which runs in kernel space with kernel mode privileges.
SIO4	This is used as a general reference to any SYNC version board supported by this driver.

1.4. Software Overview

This release of the SIO4 driver includes an executable device driver and a library of SYNC specific services. The driver software executes under control of the Linux operating system and runs in Kernel Mode as a Kernel Mode device driver. The SIO4 device driver is implemented as a standard loadable Linux device driver written in the 'C' programming language. With the driver, user applications are able to open and close a channel and, while open, perform read, write and I/O control operations. SIO4-SYNC applications must link the SYNC library with their application in order to utilize the additional, library-based services.

1.5. Hardware Overview

NOTE: The SIO8 boards appear to the driver as two SIO4 boards.

The SIO4 is a four-channel high-speed serial interface I/O board. This board provides for bi-directional serial data transfers between two computers, or one computer and an external peripheral.

This board also can transfer data indefinitely without host intervention. Once the data link between the two computers is established, the desired transfers can be performed and will become transparent to the user.

The SIO4 board includes a DMA controller and comes with a maximum of 256K Bytes of FIFO storage, which is 32K per channel side (32K * 2 * 4). The FIFO configuration can vary greatly from one SIO4 version to another (i.e., 32K * 2 * 4 to 1K * 2 *1 to none at all). The SIO4 includes configurable transceiver support that includes RS-232 and RS-485/422 transceiver options. The DMA controller is capable of transferring data to and from host memory; whereas the FIFO memory provides a means for continuous transfer of data without interrupting the DMA transfers or requiring intervention from the host CPU. The board also provides for interrupt generation for various states of the board like FIFO empty, FIFO full and DMA complete.

1.6. Reference Material

The following reference material may be of particular benefit in using the SIO4 and its device driver. The specifications provide the information necessary for an in depth understanding of the specialized features implemented on this board.

- The applicable *SIO4/SIO8 User Manual* from General Standards Corporation.
- The *PCI Bus Master Interface Chip* data handbook for the PCI9056/9080 from PLX Technology, Inc.

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

1.7. Licensing

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

2. Installation

2.1. CPU and Kernel Support

The driver is designed to operate with Linux kernel versions 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	X86	
		32-bit	64-bit
4.18.16	Red Hat Fedora Core 29	Yes	Yes
4.16.3	Red Hat Fedora Core 28	Yes	Yes
4.13.9	Red Hat Fedora Core 27		Yes
4.11.8	Red Hat Fedora Core 26	Yes	Yes
4.8.6	Red Hat Fedora Core 25	Yes	Yes
4.5.5	Red Hat Fedora Core 24	Yes	Yes
4.2.3	Red Hat Fedora Core 23	Yes	Yes
4.0.4	Red Hat Fedora Core 22	Yes	Yes
3.11.10	Red Hat Fedora Core 20	Yes	Yes
3.9.5	Red Hat Fedora Core 19	Yes	Yes
3.6.10	Red Hat Fedora Core 18	Yes	Yes
3.3.4	Red Hat Fedora Core 17	Yes	Yes
3.1.0	Red Hat Fedora Core 16	Yes	Yes
2.6.38	Red Hat Fedora Core 15	Yes	Yes
2.6.35	Red Hat Fedora Core 14	Yes	Yes
2.6.33	Red Hat Fedora Core 13	Yes	Yes
2.6.31	Red Hat Fedora Core 12	Yes	Yes
2.6.29	Red Hat Fedora Core 11	Yes	Yes
2.6.27	Red Hat Fedora Core 10	Yes	Yes
2.6.25	Red Hat Fedora Core 9	Yes	Yes
2.6.23	Red Hat Fedora Core 8	Yes	Yes
2.6.21	Red Hat Fedora Core 7	Yes	Yes
2.6.18	Red Hat Fedora Core 6	Yes	Yes
2.6.15	Red Hat Fedora Core 5	Yes	Yes
2.6.11	Red Hat Fedora Core 4	Yes	Yes
2.6.9	Red Hat Fedora Core 3	Yes	Yes

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 rebuilt before being used as the driver 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 on an SMP host.

2.2. Compiler Support

The 32-bit build for this driver relies on the use of the GCC compiler. This dependence is due only to the driver's use of the file `divdi3.c`, which is copied from GCC 2.95.1. The driver build process has been verified according to the above CPU and kernel support paragraph. The build process may fail under other build environments.

NOTE: The dependence on the GCC compiler is due to the driver's use of 64-bit integer division. This division is performed during configuration of the programmable oscillator present on some versions of the SIO4. Under the 2.2 and 2.4 kernels the needed library services are linked implicitly during the build process. Under the driver build process for the 2.6 and later kernels, the needed services must be linked explicitly.

2.3. The /proc/ File System

NOTE: All SIO8 model boards appear as two SIO4 model boards.

While the driver is running, the `/proc/sio4` file 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 then the entry value. Below is an example of what appears in the file, followed by descriptions of each entry.

```
version: 1.59.104.47
32-bit support: yes (native)
boards: 1
models: SIO4BX-SYNC
ids: 0x3
```

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 "`yes (native)`" for 32-bit installations and "`no`" for 64-bit installations.
`boards`	This identifies the total number of SIO4 boards the driver detected.
`models`	This is a list that identifies the basic model numbers of the boards detected by the driver. The order in the list corresponds to the device node indexes in the `/dev/` directory. If the driver cannot specifically identify the board's type it will be listed only as "SIO4".
`ids`	This is a list identifying the values read from the boards' user jumpers. This will be given in the C form of `printf("0x%lX", value)`. For SIO8 model boards this will be given in the C form of `printf("0x%lX.%ld", value, index)`, where *index* is the zero-based index of the SIO4 on that board. Examples are `0xF` and `0xF.0`, respectively.

2.4. File List

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

File	Description
`sio4.tar.gz`	This archive contains the driver, the API Library and all related files.
`sio4_sync_linux_um.pdf`	This is a PDF version of this user manual, which is included in the archive.

2.5. Directory Structure

The following table describes the directory structure observed by the source archive.

Directory	Content
`…/sio4/`	This is the driver root directory. It contains the documentation, the Overall Make Script (section 2.8, page 12) and the below listed subdirectories.
`…/async/`	This directory contains the Asynchronous serial protocol support files.
`…/async/lib/`	This directory contains the Asynchronous library sources.
`…/async/samples/`	This directory contains the Asynchronous specific sample applications.
`…/async/samples/asyncc2c/`	This directory contains the Asynchronous `asyncc2c` sample application.

…/async/samples/rxasync/	This directory contains the Asynchronous `rxasync` sample application.
…/async/samples/txasync/	This directory contains the Asynchronous `txasync` sample application.
…/docsrc/	This directory contains the code samples from this document (section 3.4.1, page 14).
…/driver/	This directory contains the driver and its sources (section 4, page 15).
…/async/include/	This directory contains the driver, library and utility interface header files.
…/async/lib/	This directory contains the driver, library and utility static libraries.
…/samples/	This directory contains sample applications.
…/samples/id/	This directory contains the `id` sample application.
…/samples/irq/	This directory contains the `irq` sample application.
…/samples/regs/	This directory contains the `regs` sample application.
…/samples/sbtest/	This directory contains the `sbtest` sample application.
…/sync/	This directory contains the SYNC model SIO4 support files.
…/sync/docsrc/	This directory contains the SYNC document code samples.
…/sync/lib/	This directory contains the SYNC model SIO4 library sources.
…/sync/samples/	This directory contains the SYNC specific sample applications.
…/sync/samples/syncc2c/	This directory contains the SYNC `syncc2c` sample application.
…/sync/samples/txrate/	This directory contains the SYNC `txrate` sample application.
…/utils/	This directory contains utility sources used by the sample applications.

2.6. 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 `sio4.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 structure described earlier and copies all of the archive files into the created directories.

```
tar –xzvf sio4.tar.gz
```

2.7. Removal

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

1. Shutdown the driver as described in previous paragraphs.

2. Change to the directory where the driver archive was installed. This should be `/usr/src/linux/drivers/`.

3. Issue the below command to remove the driver archive and all of the installed driver files.

```
rm –rf sio4.tar.gz sio4
```

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

```
rm –f /dev/sio4*
```

5. If the automated startup procedure was adopted, then edit the system startup script `rc.local` and remove the line that invokes the `start` script. The file `rc.local` should be located in the `/etc/rc.d/` directory.

2.8. Overall Make Script

The 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 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/sio4/`.

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

```
./make_all
```

3. Important Support Files

3.1. Main Header File

The SIO4 driver package provides a main header file that does an include of all application level SIO4 headers. Throughout this document references are given for a variety of SIO4 specific header files. Plus, these collectively include numerous others not specifically named in this document, but which are also included in the SIO4 driver package. For ease of use it is suggested that applications include only the main header file shown below rather than individually including those headers identified separately throughout this document. Including the main header file pulls in all other pertinent SIO4 specific header files. Therefore, sources may include only this one SIO4 header and make files may reference only this one SIO4 include directory. All SIO4 API Library headers, all Protocol Library headers, and all affiliated headers are included via this one header file and are all located in this one include directory.

File	Location
`sio4_main.h`	`…/sio4/include/`

3.2. Main Library File

The SIO4 driver package provides a main statically linkable library that is a substitute for separately linking all static libraries built individually as a part of the driver package. Throughout this document references are given for a variety of SIO4 specific static libraries, though this is not all driver package created libraries. For ease of use it is suggested that applications link only the main static library file shown below rather than individually linking the entire set of SIO4 static libraries. Linking the main library file pulls in all other pertinent SIO4 specific static libraries. Therefore, make files may link only this one SIO4 library and may reference only this one SIO4 library directory. The SIO4 API Library file, all Protocol Library files, and all affiliated libraries are incorporate via this one static library file and all are located in this one library directory.

File	Location
`sio4_main.a`	`…/sio4/lib/`

3.2.1. System Libraries

In addition to linking the static library named above, 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`

3.3. Protocol Libraries

The protocol libraries provide application interfaces that are tailored to the chosen serial communications protocol. This allows one to focus on use of the protocol rather than the extensive features of the SIO4. Each protocol library implements a small set of function calls that are library specific. In addition, each protocol library implements a small number of data structures designed around the specific protocol and the underlying SIO4 hardware. This allows a user to focus on the use of the protocol rather than on configuring numerous IOCTL services individually, especially when their use may be order dependent or not applicable. The protocol libraries are bundled in their entirety with the driver package. This includes source code and affiliated files for the statically linked protocol libraries, utility code, samples and documentation. The table below summarizes the protocol library files.

Description	Files	Location
Asynchronous	*.c, …	…/sio4/async/lib/
	sio4_async.h	…/sio4/include/
	sio4_async.a	…/sio4/lib/
SYNC	*.c, …	…/sio4/sync/lib/
	sio4_sync.h	…/sio4/include/
	sio4_sync.a	…/sio4/lib/

3.4. Utility Libraries

3.4.1. Document Source Code Examples

The source code examples given in this document are provided as C files included with the driver package. This is done to verify that the code compiles correctly. Additionally, the sources are compiled and linked into a static library to simplify use of the examples. The pertinent files are identified in the following tables.

Description	Files	Location
Source Files	*.c, …	…/sio4/docsrc/
Header File	sio4_dsl.h	…/sio4/include/
Library File	sio4_dsl.a	…/sio4/lib/

SYNC	Files	Location
Source Files	*.c, …	…/sio4/sync/docsrc/
Header File	sio4_sync_dsl.h	…/sio4/include/
Library File	sio4_sync_dsl.a	…/sio4/lib/

3.4.2. Utility Source Code

Additional utility sources are provided, which are also designed to aid in the understanding and use the SIO4. The essence of these utilities is to implement visual wrappers around the corresponding service. The utility services are used by the sample applications. The utility sources are compiled and linked into static libraries to simplify their use. The pertinent files are identified in the following table.

Description	Files	Location
Source Files	*.c, …	…/sio4/utils/
Header File	sio4_utils.h	…/sio4/include/
Library File	sio4_utils.a	…/sio4/lib/

3.5. Sample Applications

The driver package includes several example applications. These may be useful both for testing and for programming demonstration purposes. The examples make extensive use of the utility libraries also included in the driver package. The files are located as given in the table below.

Description	Location
Generic	…/sio4/samples/
Asynchronous	…/sio4/async/samples/
SYNC	…/sio4/sync/samples/

4. The Driver

The paragraphs that follow give instructions on building, starting and verifying startup of the driver. These files are installed into the `/usr/src/linux/drivers/sio4/driver/` directory.

> **NOTE:** This driver works with both SYNC and non-SYNC versions of the SIO4. The driver used here is the same exact driver provided with the non-SYNC driver release.

File	Description
`*.c`	These sources implement the driver interface and its functionality. Some functionality has been modularized based on individual source file base names.
`*.h`	These are driver header files. Others are listed below.
`Makefile`	This is the driver make file.
`makefile.dep`	This is a make dependency file. This is updated automatically.
`sio4.h`	This is the driver interface header file. It should be included by SIO4 applications.
`start`	This is a shell script to install the driver module and device nodes.

4.1. Build

Follow the below steps to build the driver.

1. Change to the directory where the driver and its sources were installed. This should be `/usr/src/linux/drivers/sio4/driver/`.

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

   ```
   make clean
   ```

3. Build the driver by issuing the below command.

   ```
   make all
   ```

   > **NOTE:** Building the SIO4 driver requires installation of the kernel header sources. If they are not present the build will fail.

   > **NOTE:** Due to the differences between the many Linux distributions some build errors may occur. These errors may include system header location differences and should be easily correctable. Other errors may also appear as some distributors port newer kernel changes into older kernel distributions.

4.2. Startup

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.

4.2.1. Manual Driver Startup Procedures

Start the driver manually by following the below listed steps.

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

2. Change to the directory where the driver was installed. This should be `/usr/src/linux/drivers/sio4/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: While loading the amount of time taken for driver initialization will vary depending on the number of boards and each board's type. For those boards with programmable oscillators, additional initialization time may be needed for programming of each channel.

   NOTE: The script's default specifies that the driver is installed in the same directory as the script. The script will fail if this is not so.

   NOTE: The above step must be repeated each time the host is rebooted.

   NOTE: The SIO4 device node major number is assigned dynamically by the kernel. The minor numbers and the device node suffix numbers are index numbers beginning with one, and increase by one for each additional serial channel.

4. Verify that the device module has been loaded by issuing the below command and examining the output. The module name `sio4` 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 four nodes for each installed board.

   ```
   ls –l /dev/sio4*
   ```

4.2.2. 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/sio4/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.

4.2.2.1. 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.
```

4.2.2.2. 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.

4.2.2.3. `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/`.

4.2.2.4. `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).

4.2.2.5. 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
```

4.3. Verification

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

1. Issue the below command to view the content of the driver's `/proc/` file system text file.

   ```
   cat /proc/sio4
   ```

2. If the file exists then the driver is installed and running.

4.4. Version

The driver version number can be obtained in a variety of ways. It is reported by the driver both when the driver is loaded and when it is unloaded (depending on kernel configuration options, this may be visible only in `/var/log/messages`). It is recorded in the file `/proc/sio4`. It can also be read by an application via the `SIO4_GET_DRIVER_INFO` IOCTL services.

4.5. Shutdown

Shutdown the driver following the below listed steps.

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

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

   ```
   rmmod sio4
   ```

3. Verify that the driver module has been unloaded by issuing the below command. The module name `sio4` should not be in the list.

   ```
   lsmod
   ```

5. The SYNC Library

This library provides applications access to services and functionality specific to SIO4-SYNC boards. The driver archive includes all of the library's sources and make files. The library must be linked with SYNC applications in order to exercise the library's SYNC specific functionality. The source files are delivered undocumented but may be used to assist in application development and to help ease the learning curve. The files are described here only briefly, though library use is described in the following paragraph. The files are installed into the …/sio4/sync/lib/ directory. The files included are listed below.

File	Description
*.c	These are the sources that implement the library's functionality.
makefile	This is the library make file.
makefile.dep	This is a make dependency file. This is updated automatically.
sio4_sync.h	This is a header file that exports the library's interface. This must be included by applications using the library.

5.1. Build

Follow the below steps to compile the example files.

1. Change to the directory where the library source files were installed (…/sio4/sync/lib/).

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

   ```
   make clean
   ```

3. Build the library issuing the below command.

   ```
   make all
   ```

5.2. Use

Use of the library has compile and link time requirements. Compile time use requires inclusion of the library's header file and expansion of the include search path. The header file is named sio4_sync.h and the path to include is …/sio4/include/. Link time use requires linking two additional libraries and expansion of the library search path. The first library is the SYNC library, which is named sio4_sync.a and is located in …/sio4/lib/. The second library is the primary document source code example library, which is named sio4_sync_dsl.a and is also located in …/sio4/lib/. This library is described below.

6. Driver Interface

The SIO4 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 the GSC SIO4 board for Linux applications. The interface includes various macros, data types and functions, all of which are described in the following paragraphs. The SIO4 specific portion of the driver interface is defined in the header files `sio4.h` and `sio4_sync.h`, portions of which are described in this section. The headers define numerous items in addition to those described here.

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

> **NOTE:** The driver included with this release is designed to work with the SIO4 models listed on the cover page of this user manual, as well as other models not listed. The driver interface may therefore include IOCTL services and support components intended for use with other models. Services and support components not documented in this manual should therefore not be used with the models listed on the cover. For other SIO4 models, refer to the applicable driver user manual.

6.1. Macros

The driver interface includes the following macros which are defined in `sio4.h` and `sio4_sync.h`. These headers contain numerous additional utility type macros in addition to those described here.

6.1.1. IOCTL

The IOCTL macros are documented following the function call descriptions.

6.1.2. Registers

The following table gives the complete set of SIO4-SYNC registers. The tables are divided by register categories.

6.1.2.1. GSC Registers

The following table gives the complete set of GSC specific SIO4-SYNC registers. For detailed definitions of these registers refer to the relevant SIO4-SYNC User Manual. Please note that the set of registers supported by any given board may vary according to model and firmware version. For the set of supported registers and detailed definitions of these registers please refer to the appropriate *SIO4 User Manual*.

Macros	Description
SIO4_GSC_BCR	Board Control Register
SIO4_GSC_BSR	Board Status Register
SIO4_GSC_CCR	Clock Control Register
SIO4_GSC_CSR	Control/Status Register
SIO4_GSC_FCR	FIFO Count Register
SIO4_GSC_FDR	FIFO Data Register
SIO4_GSC_FR	Features Register
SIO4_GSC_FRR	Firmware Revision Register
SIO4_GSC_FSR	FIFO Size Register
SIO4_GSC_FTR	Firmware Type Register
SIO4_GSC_GPIOSR	GPIO Status Register (older boards)
SIO4_GSC_ICR	Interrupt Control Register
SIO4_GSC_IOCR	I/O Control Register (older boards)
SIO4_GSC_ISR	Interrupt Status Register
SIO4_GSC_IELR	Interrupt Edge/Level Register

SIO4_GSC_IHLR	Interrupt Hi/low Register
SIO4_GSC_PCDR	Programmable Clock/Divider Register (older boards)
SIO4_GSC_PCR	Programmable Clock Register (older boards)
SIO4_GSC_POCSR	Programmable Oscillator Control/Status Register
SIO4_GSC_PORAR	Programmable Oscillator RAM Address Register
SIO4_GSC_PORDR	Programmable Oscillator RAM Data Register
SIO4_GSC_PORD2R	Programmable Oscillator RAM Data 2 Register
SIO4_GSC_PSRCR	Pin Source Register
SIO4_GSC_PSTSR	Pin Status Register
SIO4_GSC_RAR	Receiver Almost Empty/Full Register
SIO4_GSC_RCR	Rx Count Register
SIO4_GSC_SBR	Sync Byte Register
SIO4_GSC_TAR	Transmitter Almost Empty/Full Register
SIO4_GSC_TCR	Tx Count Register
SIO4_GSC_TSR	Timestamp Register

6.1.2.2. PCI Configuration Registers

Access to the PCI registers is seldom required so these registers are not listed here. For the complete list of the PCI register identifiers refer to the driver header file `sio4.h`.

6.1.2.3. 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 PCI9056 and PCI9080 register identifiers refer to the driver header file `sio4.h`.

6.2. Data Types

This driver interface includes the following data types which are defined in `sio4.h`.

6.2.1. FIFO_STATUS

This enumeration defines various possible values that may be received when reading a FIFO's status.

> **NOTE:** Other values are possible but are not seen in normal use.
>
> **NOTE:** The Almost Empty status becomes active when the FIFO contains *ALMOST EMPTY* or fewer bytes. Here, *ALMOST EMPTY* refers to the value programmed into the lower 16 bits of the Tx and Rx Almost Registers.
>
> **NOTE:** The Almost Full status becomes active when the FIFO can receive *ALMOST FULL* or fewer additional bytes before being full. Here, *ALMOST FULL* refers to the value programmed into the upper 16 bits of the Tx and Rx Almost Registers.

Definition

```
typedef enum FIFOStatus
{
    …
} FIFO_STATUS;
```

Values	Description
ALMOST_EMPTY_STATUS	The FIFO is almost full.
ALMOST_FULL_STATUS	The FIFO is almost full.

EMPTY_STATUS	The FIFO is empty.
FULL_STATUS	The FIFO is full.
INVALID_STATUS	The FIFO's current status is invalid.
NOT_ALMOST_EMPTY_NOR _ALMOST_FULL_STATUS	The FIFO level is between the almost full and the almost empty states.

6.2.2. sio4_driver_info_t

This structure defines the data fields for the information returned by the SIO4_GET_DRIVER_INFO IOCTL service.

Definition

```
typedef struct SIO4DriverInfo
{
    __u8    version[8];
    __u8    built[32];
} sio4_driver_info_t;
```

Fields	Description
version	This field gives the driver version number as a string in the form of X.X.X.X.
built	The driver no longer provides its build data and time, so this field will be empty.

6.2.3. SIO4_INTERRUPT_STATUS

This structure records the interrupt status bits from the SIO4 Interrupt Status Register for the current channel. The bits reflect the accumulated status since the last interrupt notification or status request.

Definition

```
typedef struct IntStatus
{
    __u8    u8SIO4Status;
} SIO4_INTERRUPT_STATUS;
```

Fields	Description
u8SIO4Status	The channel's interrupt status from the Interrupt Status Register. This may consist of either four or eight bits, depending on the board's capabilities.

6.2.4. sio4_mp_chip_t

This enumeration identifies the supported options for identifying the Multi-Protocol transceiver feature on the SIO4. The values are used in the chip field of the sio4_mp_t (section 6.2.6, page 23) data structure, which is used with the Multi-Protocol transceiver based IOCTL services. Refers to the specific service for information on how this structure is used.

Definition

```
typedef enum
{
    …
} sio4_mp_chip_t;
```

Values	Description
SIO4_MP_CHIP_FIXED	This refers to a fixed protocol implementation. The driver may not know which protocol is implemented on the SIO4.
SIO4_MP_CHIP_SP508	This refers to the Sipex SP508 Multi-Protocol transceiver chip.
SIO4_MP_CHIP_UNKNOWN	The chip type is unknown.

6.2.5. sio4_mp_prot_t

This enumeration identifies the protocol options supported by the Multi-Protocol transceiver driver. The values are used in the `want` and `got` fields of the `sio4_mp_t` (section 6.2.6, page 23) data structure, which is used with the Multi-Protocol transceiver based IOCTL services. Refers to the specific service for information on how this structure is used. Refer to the hardware user manual for detailed explanations of each protocol options.

Definition

```
typedef enum
{
    …
} sio4_mp_prot_t;
```

Values	Description
SIO4_MP_PROT_RS_232	This refers to the RS-232 protocol.
SIO4_MP_PROT_RS_422_485	This refers to the RS-422/RS-485 protocol.
SIO4_MP_PROT_RS_422_423_MM1	This refers to the RS-422/RS-423, Mixed Mode 1 protocol.
SIO4_MP_PROT_RS_422_423_MM2	This refers to the RS-422/RS-423, Mixed Mode 2 protocol.
SIO4_MP_PROT_RS_423	This refers to the RS-423 protocol.
SIO4_MP_PROT_RS_530_M1	This refers to the RS-530, Mode 1 protocol.
SIO4_MP_PROT_RS_530_M2	This refers to the RS-530, Mode 2 protocol.
SIO4_MP_PROT_V35_M1	This refers to the V.35, Mode 1 protocol.
SIO4_MP_PROT_V35_M2	This refers to the V.35, Mode 2 protocol.
SIO4_MP_PROT_DISABLE	This refers to the disabled or tri-stated condition.
SIO4_MP_PROT_INVALID	This is returned by the driver when a requested protocol is unsupported or unrecognized.
SIO4_MP_PROT_READ	This requests that the driver report the current protocol.
SIO4_MP_PROT_UNKNOWN	This is returned by the driver when the protocol is unknown.

6.2.6. sio4_mp_t

This data structure is used to exchange information and requests about the board's Multi-Protocol transceiver feature between applications and the driver. This structure is used with the Multi-Protocol transceiver based IOCTL services. Refers to the specific service for information on how this structure is used.

Definition

```
typedef struct
{
    __s32   chip;
    __s32   prot_want;
    __s32   prot_got;
} sio4_mp_t;
```

Field	Description
chip	The driver will fill this field in with the Multi-Protocol transceiver chip identifier. Refer to the sio4_mp_chip_t (section 6.2.4, page 22) data type documentation elsewhere in this document.
prot_want	This refers to the protocol desired by the application.
prot_got	This refers to the protocol reported by the device.

6.2.7. sio4_osc_chip_t

This enumeration identifies the supported options for identifying the programmable oscillator feature on the SIO4. The values are used in the chip field of the sio4_osc_t (section 6.2.8, page 24) data structure, which is used with the programmable oscillator based IOCTL services. Refers to the specific service for information on how this structure is used.

Definition

```
typedef enum
{
    …
} sio4_osc_chip_t;
```

Values	Description
SIO4_OSC_CHIP_CY22393	This refers to a single Cypress CY22393, which provides each SIO4 channel with its own programmable oscillator.
SIO4_OSC_CHIP_CY22393_2	This refers to two Cypress CY22393s, which provides each SIO4 channel with its own programmable oscillator.
SIO4_OSC_CHIP_FIXED	This refers to a fixed frequency, non-programmable oscillator that is shared by all SIO4 channels.
SIO4_OSC_CHIP_IDC2053B	This refers to a single Cypress IDC2053B, which provides all SIO4 channel with the same programmable oscillator output.
SIO4_OSC_CHIP_IDC2053B_4	This refers to four Cypress IDC2053B programmable oscillators, which provides each SIO4 channel with its own output.
SIO4_OSC_CHIP_UNKNOWN	The oscillator is unknown.

6.2.8. sio4_osc_t

This data structure is used to exchange information and requests about the board's programmable oscillator between applications and the driver. This structure is used with the programmable oscillator based IOCTL services. Refers to the specific service for information on how this structure is used.

Definition

```
typedef struct
{
    __u32   chip;
    __s32   freq_ref;
    __s32   freq_want;
    __s32   freq_got;
} sio4_osc_t;
```

Field	Description
chip	The driver will fill this field in with the oscillator chip identifier. Refer to the sio4_osc_chip_t (section 6.2.7, page 24) data type documentation elsewhere in this document.
freq_ref	This refers to the frequency of the oscillator's reference source.

freq_want	This refers to the clock output frequency desired by the application.
freq_got	This refers to the clock output frequency produced by the device.

6.2.9. sio4_reg_t

This structure defines the data fields applicable to performing register read, write, and read-modify-write operations with the register access IOCTL services.

Definition

```
typedef struct
{
    __u32   reg;
    __u32   value;
    __u32   mask;
} sio4_reg_t;
```

Fields	Description
reg	This identifies the register to access.
value	The register value is placed here. This is either the value read from the register, the value to write to the register, or the bits to apply for modifications.
mask	This is the set of bits to modify for a read-modify-write access. If a bit is set here, then the corresponding "value" bit is applied to the register. Otherwise, the register bit is unmodified.

6.2.10. sio4_sync_rx_t

This structure defines SYNC specific receiver data fields. It is used both for retrieving configuration settings and applying configuration settings.

NOTE: This data type is defined in sio4_sync.h.

Definition

```
typedef struct
{
    int     enable;
    int     lsbf;
    int     reset;
    __u16   word_size;

    struct
    {
        int high;
    } clock;

    struct
    {
        int enable;
        int high;
    } env;

    struct
    {
        int high;
    } data;
```

```
} sio4_sync_rx_t;
```

Fields	Description
enable	This refers to the enabled/disabled state of the receiver. If zero, then the receiver is disabled. If non-zero, then the receiver is enabled.
lsbf	This refers to the bit order that data is received in. If zero, then data is clocked in Most Significant Bit First. If non-zero, then data is clocked in Least Significant Bit First, hence the field name lsbf.
reset	This refers to Rx Count errors. If this is reported as zero, then no errors have occurred. If non-zero an error has occurred and needs to be reset. If zero when settings are being applied, then no action is taken. If non-zero then the error is reset.
word_size	This refers to the number of bits in the last received word.
clock	This refers to the Rx Clock configuration.
clock.high	If zero, then the Rx Clock is active low (falling edge). If non-zero, then it is active high (rising edge).
env	This refers to the Rx Envelope configuration.
env.enable	This refers to enabling or disabling the use of the Rx Envelope. If zero, then it is disabled. This is applicable for two-wire configurations. If non-zero, then it is enabled. This is applicable to three-wire configurations.
env.high	If zero, then Rx Envelope is active low. If non-zero, then it is active high. This is ignored when Rx Envelope is disabled.
data	This refers to the Rx Data configuration.
data.high	If zero, then Rx Data is active low. If non-zero, then it is active high. This is ignored when Rx Data is disabled.

6.2.11. sio4_sync_t

This structure defines SYNC specific data fields that are independent of the board's transmitter and receiver. It is used both for retrieving configuration settings and applying configuration settings.

NOTE: This data type is defined in sio4_sync.h.

Definition

```
typedef struct
{
    int dce_enable;
    int dte_enable;

    struct
    {
        int enable;
        int internal;
    } lb;
} sio4_sync_t;
```

Fields	Description
dce_enable	This refers to the board's DCE configuration. If zero, then the DCE mode is disabled. If non-zero, then the DCE mode is enabled. *
dte_enable	This refers to the board's DTE configuration. If zero, then the DTE mode is disabled. If non-zero, then the DTE mode is enabled. The SYNC library ignores this field if the dce_enable field is non-zero. *
lb	This substructure refers to the board's loopback configuration.
enable	This refers to the board's signal loopback capability. If zero, then the loopback feature is disabled. If non-zero, then the feature is enabled.

`internal`	This refers to the use of internal or external loopback. If zero, then external loopback is utilized. If non-zero, then internal loopback is used. This field is ignored when loopback is disabled.

* Either the DCE mode or the DTE mode must be enabled for data to be transferred across the cable interface. If both are disabled, then no data can be transferred.

6.2.12. sio4_sync_tx_t

This structure defines SYNC specific transmitter data fields. It is used both for retrieving configuration settings and applying configuration settings.

NOTE: This data type is defined in `sio4_sync.h`.

Definition

```
typedef struct
{
    int     enable;
    int     auto_dis;
    int     lsbf;
    __u16   word_size;
    __u16   gap_size;

    struct
    {
        int clock;
        int ext;
        int high;
        int idle;
    } clock;

    struct
    {
        int env;
        int high;
    } env;

    struct
    {
        int data;
        int high;
    } data;

    struct
    {
        int enable;
        int clock;
        int high;
    } aux_clock;

    struct
    {
        int enable;
        int high;
    } spare;
} sio4_sync_tx_t;
```

Fields	Description
enable	This refers to the enabled/disabled state of the transmitter. If zero, then the transmitter is disabled. If non-zero, then the transmitter is enabled.
auto_dis	This refers to automatic disabling of the transmitter when the Tx FIFO runs empty. If zero, then the transmitter is not automatically disabled when the Tx FIFO runs empty. If non-zero, then the transmitter is automatically disabled when the Tx FIFO runs empty.
lsbf	This refers to the bit order that data is transmitted. If zero, then data is clocked out Most Significant Bit First. If non-zero, then data is clocked out Least Significant Bit First, hence the field name lsbf. This refers to each byte within each transmit word. Multi-byte words are always sent out in the order that appear in the Tx FIFO.
word_size	This refers to the size of each transmit word in bits.
gap_size	This refers to the number of bits transmitter at the completion of each transmit word.
clock	This refers to the Tx Clock configuration.
clock.clock	This refers to the functionality of the Tx Clock signal. If zero, then the signal is a GPIO Output. If non-zero, then it is the transmit clock.
clock.ext	This refers to the transmit clock's input source. If zero, then it is the onboard oscillator divided by two. If non-zero, then it is the Rx Auxiliary Clock.
clock.high	If Tx Clock is functioning as the transmit clock, then it is active low (falling edge) when zero, and active high (rising edge) when non-zero. For the GPIO functionality, it is low when zero and high when non-zero.
clock.idle	This refers to the transmit clock's operation when the transmitter is idle. When zero, the signal is not idle. It is driven with the transmit clock. When non-zero the signal is idle and is driven low.
env	This refers to the Tx Envelope configuration.
env.env	This refers to the functionality of the Tx Envelope signal. If zero, then the signal is a GPIO Output. If non-zero, then it is the transmit envelope.
env.high	If Tx Envelope is functioning as the transmit envelope, then it is active low when zero, and active high when non-zero. For the GPIO functionality, it is low when zero and high when non-zero.
data	This refers to the Tx Data configuration.
data.data	This refers to the functionality of the Tx Data signal. If zero, then the signal is a GPIO Output. If non-zero, then it is the transmit data.
data.high	If Tx Data is functioning as the transmit data, then it is active low when zero, and active high when non-zero. For the GPIO functionality, it is low when zero and high when non-zero.
aux_clock	This refers to the Tx Auxiliary Clock configuration.
aux_clock.enable	This refers to the enable/disable state of the signal. If zero, then it is disabled and tri-stated. If non-zero, then it operates according to the below settings.
aux_clock.clock	This refers to the functionality of the Tx Auxiliary Clock signal. If zero, then the signal is a GPIO Output. If non-zero, then it is driven with the onboard oscillator divided by two.
aux_clock.high	If Tx Auxiliary Clock is functioning as a clock output, then it is active low (falling edge) when zero, and active high (rising edge) when non-zero. For the GPIO functionality, it is low when zero and high when non-zero.
spare	This refers to the Tx Spare configuration.
spare.enable	This refers to the enable/disable state of the signal. If zero, then it is disabled and tri-stated. If non-zero, then it is a GPIO output.
spare.high	If Tx Spare is enabled, then it is low when zero and high when non-zero.

6.2.13. TX_RX

This enumeration defines the possible external FIFO reset selection options.

Definition

```
typedef enum TxRx
{
    …
} TX_RX;
```

Fields	Description
RX_FIFO	Reset the receive FIFO.
TX_FIFO	Reset the transmit FIFO.
TX_AND_RX_FIFO	Reset both FIFOs.

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 SIO4 serial channel. The device is put in an initialized state before this call returns. The programmable oscillator, if present, is not modified.

> **NOTE**: This call does not change the firmware type on those boards whose firmware type is configurable.

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 <stdio.h>

#include "sio4_dsl.h"

int sio4_close(int fd)
{
    int errs;
    int ret;

    ret = close(fd);

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

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

6.3.2. ioctl()

This function is the entry point to performing setup and control operations on an open SIO4 serial channel. 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 a following section.

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 SIO4 IOCTL services use at most one argument, which is represented by a 32-bit value.

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

Example

```
#include <stdio.h>

#include "sio4_dsl.h"

int sio4_ioctl(int fd, int request, void* arg)
{
    int errs;
    int ret;

    ret = ioctl(fd, request, arg);

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

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

6.3.3. open()

This function is the entry point to open a connection to an SIO4 serial channel. The device is initialized before the function returns. The programmable oscillator, if present, is not modified.

NOTE: The SIO8 appears to the driver as two SIO4 boards.

NOTE: This call does not change the firmware type on those boards whose firmware type is configurable.

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 <stdio.h>

#include "sio4_dsl.h"

int sio4_open(int device, int* fd)
{
    int     errs    = 0;
    char    name[128];
    int     ret;

    sprintf(name, "/dev/sio4%d", device);
    ret = open(name, O_RDWR);

    if (ret < 0)
    {
        printf("ERROR: open() returned %d\n", ret);
        errs    = 1;
    }

    if (fd)
        fd[0]   = ret;


return(errs);

}
```

6.3.4. read()

This function is the entry point to reading received data from an open SIO4 serial channel. This function should only be called after a successful open of the respective device. The function reads up to bytes bytes from the receive FIFO. If the number of bytes requested is not available within the configured time limit, the read operation times out.

NOTE: Refer to the `SIO4_RX_IO_MODE_CONFIG` IOCTL services to configure this call for use of PIO, Block Mode DMA or Demand Mode DMA data transfer.

Prototype

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

Argument	Description
`fd`	This is the file descriptor of the device to access.
`buf`	The data read will be put here.
`bytes`	This is the desired number of bytes to read.

Return Value	Description
`-1`	An error occurred. Consult `errno`.
`0` to `bytes`	The operation succeeded. If the return value is less than `bytes`, then the request timed out.

Example

```
#include <stdio.h>

#include "sio4_dsl.h"

int sio4_read(int fd, void* dst, size_t bytes, size_t* qty)
{
    int errs;
    int ret;

    ret = read(fd, dst, bytes);

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

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


errs = (ret < 0) ? 1 : 0;

return(errs);

}
```

6.3.5. sio4_sync_get()

This function is the entry point to retrieving SYNC configuration settings that are independent of the transmitter and receiver. This function should only be called after a successful open of the respective device.

NOTE: The prototype for this function is in `sio4_sync.h`.

Prototype

```
int sio4_sync_get(int fd, sio4_sync_t* sync);
```

Argument	Description
fd	This is the file descriptor of the device to access.
sync	This points to the structure that is to receive the current configuration state (section 6.2.11, page 26). This pointer may be NULL.

Return Value	Description
0	No errors were encountered.
> 0	An appropriate error value, which is the applicable errno value.

Example

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

#include "sio4_sync_dsl.h"

int sio4_sync_dce_set(int fd, int enable, int verbose)
{
    sio4_sync_t sync;
    int         status;
    const char* str = "sio4_sync_get";

    status  = sio4_sync_get(fd, &sync);

    if (status == 0)
    {
        sync.dce_enable= enable;
        status         = sio4_sync_set(fd, &sync);
        str            = "sio4_sync_set";
    }

    if ((verbose) && (status == -1))
        printf("%s() failure, errno = %d\n", str, errno);

    return(status);
}
```

6.3.6. sio4_sync_gpio_rx()

This function is the entry point to reading the SYNC board's GPIO inputs. The value read represents the current value of all signals included in the Pin Source Register, even those not configured as GPIO. Bits undefined in the register are returned as zero. This function should only be called after a successful open of the respective device.

NOTE: The prototype for this function is in sio4_sync.h.

Prototype

```
int sio4_sync_gpio_rx(int fd, __u32* value);
```

Argument	Description
fd	This is the file descriptor of the device to access.
value	This points to the variable to receive the value read. This pointer may be NULL.

Return Value	Description
0	No errors were encountered.
> 0	An appropriate error value, which is the applicable errno value.

Example

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

#include "sio4_sync_dsl.h"

int sio4_sync_gpio_read_tx(int fd, __u32* value, int verbose)
{
    int status;

    status     = sio4_sync_gpio_rx(fd, value);
    value[0]   &= 0x2F0;

    if ((verbose) && (status == -1))
        printf("sio4_sync_gpio_read()   failure,   errno   =   %d\n",
errno);

    return(status);
}
```

6.3.7. sio4_sync_gpio_tx()

This function is the entry point to writing to the SYNC board's GPIO outputs. Only those signals currently configured as GPIO outputs are affected. All bits that do not correspond to GPIO outputs are quietly ignored. This function should only be called after a successful open of the respective device.

NOTE: The prototype for this function is in sio4_sync.h.

Prototype

```
int sio4_sync_gpio_tx(int fd, __u32 value);
```

Argument	Description
fd	This is the file descriptor of the device to access.
value	This is the value to write to the GPIO outputs.

Return Value	Description
0	No errors were encountered.
> 0	An appropriate error value, which is the applicable errno value.

Example

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

#include "sio4_sync_dsl.h"
```

```
int sio4_sync_gpio_mod(int fd, __u32 value, __u32 mask, int verbose)
{
    int         status;
    const char* str = "sio4_sync_gpio_rx";
    __u32       v;

    status  = sio4_sync_gpio_rx(fd, &v);

    if (status == 0)
    {
        value   = (value & mask) | (v & ~mask);
        status  = sio4_sync_gpio_tx(fd, value);
        str     = "sio4_sync_gpio_tx";
    }

    if ((verbose) && (status == -1))
        printf("%s() failure, errno = %d\n", str, errno);

    return(status);
}
```

6.3.8. sio4_sync_rx_get()

This function is the entry point to retrieving receiver specific SYNC configuration settings. This function should only be called after a successful open of the respective device.

NOTE: The prototype for this function is in `sio4_sync.h`.

Prototype

```
int sio4_sync_rx_get(int fd, sio4_sync_rx_t* rx);
```

Argument	Description
`fd`	This is the file descriptor of the device to access.
`rx`	This points to the structure that is to receive the current configuration state (section 6.2.10, page 25). This pointer may be NULL.

Return Value	Description
`0`	No errors were encountered.
`> 0`	An appropriate error value, which is the applicable `errno` value.

Example

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

#include "sio4_sync_dsl.h"

int sio4_sync_rx_data_get(int fd, int* high, int verbose)
{
    sio4_sync_rx_t  rx;
```

```
    int             status;

    status  = sio4_sync_rx_get(fd, &rx);
    high[0] = rx.data.high;

    if ((verbose) && (status == -1))
        printf("sio4_sync_rx_get() failure, errno = %d\n", errno);

    return(status);
}
```

6.3.9. sio4_sync_rx_set()

This function is the entry point to applying receiver specific SYNC configuration settings. This function should only be called after a successful open of the respective device.

NOTE: The prototype for this function is in `sio4_sync.h`.

Prototype

```
int sio4_sync_rx_set(int fd, const sio4_sync_rx_t* rx);
```

Argument	Description
`fd`	This is the file descriptor of the device to access.
`rx`	This points to the structure that contains the settings to apply (section 6.2.10, page 25). This pointer may be NULL.

Return Value	Description
`0`	No errors were encountered.
`> 0`	An appropriate error value, which is the applicable `errno` value.

Example

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

#include "sio4_sync_dsl.h"

int sio4_sync_rx_env_set(int fd, int enable, int high, int verbose)
{
    sio4_sync_rx_t  rx;
    int             status;
    const char*     str = "sio4_sync_rx_get";

    status  = sio4_sync_rx_get(fd, &rx);

    if (status == 0)
    {
        rx.env.enable   = enable;
        rx.env.high     = high;
        status          = sio4_sync_rx_set(fd, &rx);
        str             = "sio4_sync_rx_set";
```

```
    }

    if ((verbose) && (status == -1))
        printf("%s() failure, errno = %d\n", str, errno);

    return(status);
}
```

6.3.10. sio4_sync_set()

This function is the entry point to applying SYNC configuration settings that are independent of the transmitter and receiver. This function should only be called after a successful open of the respective device.

NOTE: The prototype for this function is in `sio4_sync.h`.

Prototype

```
int sio4_sync_set(int fd, const sio4_sync_t* sync);
```

Argument	Description
`fd`	This is the file descriptor of the device to access.
`sync`	This points to the structure containing the setting to apply (section 6.2.11, page 26). This pointer may be NULL.

Return Value	Description
`0`	No errors were encountered.
`> 0`	An appropriate error value, which is the applicable `errno` value.

Example

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

#include "sio4_sync_dsl.h"

int sio4_sync_dte_get(int fd, int* enable, int verbose)
{
    sio4_sync_t sync;
    int         status;

    status      = sio4_sync_get(fd, &sync);
    enable[0]   = sync.dte_enable;

    if ((verbose) && (status == -1))
        printf("sio4_sync_get() failure, errno = %d\n", errno);

    return(status);
}
```

6.3.11. sio4_sync_tx_get()

This function is the entry point to retrieving transmitter specific SYNC configuration settings. This function should only be called after a successful open of the respective device.

NOTE: The prototype for this function is in `sio4_sync.h`.

Prototype

```
int sio4_sync_tx_get(int fd, sio4_sync_tx_t* tx);
```

Argument	Description
`fd`	This is the file descriptor of the device to access.
`tx`	This points to the structure that is to receive the current configuration state (section 6.2.12, page 27). This pointer may be NULL.

Return Value	Description
`0`	No errors were encountered.
`> 0`	An appropriate error value, which is the applicable `errno` value.

Example

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

#include "sio4_sync_dsl.h"

int sio4_sync_tx_env_get(int fd, int* env, int* high, int verbose)
{
    sio4_sync_tx_t  tx;
    int             status;

    status  = sio4_sync_tx_get(fd, &tx);
    env[0]  = tx.env.env;
    high[0] = tx.env.high;

    if ((verbose) && (status == -1))
        printf("sio4_sync_tx_get() failure, errno = %d\n", errno);

    return(status);
}
```

6.3.12. sio4_sync_tx_set()

This function is the entry point to applying transmitter specific SYNC configuration settings. This function should only be called after a successful open of the respective device.

NOTE: The prototype for this function is in `sio4_sync.h`.

Prototype

```
int sio4_sync_tx_set(int fd, const sio4_sync_tx_t* tx);
```

Argument	Description
fd	This is the file descriptor of the device to access.
tx	This points to the structure that contains the settings to apply (section 6.2.12, page 27). This pointer may be NULL.

Return Value	Description
0	No errors were encountered.
> 0	An appropriate error value, which is the applicable errno value.

Example

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

#include "sio4_sync_dsl.h"

int sio4_sync_tx_data_set(int fd, int data, int high, int verbose)
{
    sio4_sync_tx_t  tx;
    int             status;
    const char*     str = "sio4_sync_tx_get";

    status  = sio4_sync_tx_get(fd, &tx);

    if (status == 0)
    {
        tx.data.data    = data;
        tx.data.high    = high;
        status          = sio4_sync_tx_set(fd, &tx);
        str             = "sio4_sync_tx_set";
    }

    if ((verbose) && (status == -1))
        printf("%s() failure, errno = %d\n", str, errno);

    return(status);
}
```

6.3.13. write()

This function is the entry point to writing data for transmission to an open SIO4 serial channel. This function should only be called after a successful open of the respective device. The function writes up to bytes bytes to the transmit FIFO. If the number of bytes requested cannot be sent within the configured time limit, the write operation times out.

> **NOTE:** Refer to the SIO4_TX_IO_MODE_CONFIG IOCTL services to configure this call for use of PIO, Block Mode DMA or Demand Mode DMA data transfer.

Prototype

```
int write(int fd, const void *buf, size_t bytes);
```

Argument	Description
fd	This is the file descriptor of the device to access.
buf	The data written comes from here.
bytes	This is the desired number of bytes to write.

Return Value	Description
-1	An error occurred. Consult errno.
0 to bytes	The operation succeeded. If the return value is less than bytes, then the request timed out.

Example

```
#include <stdio.h>

#include "sio4_dsl.h"

int sio4_write(int fd, const void *src, size_t bytes, size_t* qty)
{
    int errs;
    int ret;

    ret = write(fd, src, bytes);

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

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


  errs = (ret < 0) ? 1 : 0;

  return(errs);

}
```

6.4. IOCTL Services

The SIO4 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 and is an unsigned 32-bit data type. Unless otherwise stated the return value definitions are those defined for the ioctl() function call.

> **NOTE:** Many of the IOCTL services alter the state of the channel's operation and can adversely affect the channel's proper operation if data transfer is in progress. Exercise care when using these services to ensure that data integrity is maintained.

6.4.1. SIO4_BOARD_JUMPERS

This service reads the jumper information for the user jumpers. If the jumpers are not supported on the board in use, then the returned value is the XXX_UNKNOWN macro. If the jumpers are supported, then the value returned will be from 0x0 to 0x3 for boards with two jumpers and from 0x0 to 0xF for boards with four jumpers.

Usage

ioctl() Argument	Description
request	SIO4_BOARD_JUMPERS
arg	__s32*

The table below lists the predefined macros used with this service.

Macros	Description
SIO4_BOARD_JUMPERS_UNSUPPORTED	The board jumpers are unsupported.

6.4.2. SIO4_FEATURE_TEST

This service provides information on an SIO4's feature set. To gain support information on a specific feature the corresponding macro is supplied. The value returned will be the corresponding support information, which may be the XXX_YES or XXX_NO macro or some other feature specific value. If the XXX_COUNT macro is supplied, the value returned is the number of feature options supported by the service, and should be one more that the service's XXX_LAST_INDEX macro.

Usage

ioctl() Argument	Description
request	SIO4_FEATURE_TEST
arg	__s32*

The table below lists the options used with this service.

Macros	Description
SIO4_FEATURE_BOARD_RESET	Does the Board Control Register support the Board Reset bit?
SIO4_FEATURE_BUS_SPEED	This indicates the maximum PCI bus speed the board was designed for. This should be 33 (for 33MHz) or 66 (for 66MHz).
SIO4_FEATURE_BUS_WIDTH	This indicates the PCI bus width the board was designed for. This should be 32 (for 32-bits) or 64 (for 64-bits).
SIO4_FEATURE_CHANNEL_QTY	This refers to the number of channels on the entire board and should be either four or eight.
SIO4_FEATURE_COUNT	This reports the number of features supported by the service.
SIO4_FEATURE_DEVICE_QTY	This reports the number of SIO4 products built into the board. This is one for SIO4 model boards and two for SIO8 model boards.
SIO4_FEATURE_DMDMA_SCD	Does the Board Control Register support the Single Cycle Disable bit?
SIO4_FEATURE_FIFO_SIZE_RX	This is the size of the channel's Rx FIFO, or zero if the size is unknown.
SIO4_FEATURE_FIFO_SIZE_TOTAL	This is the total combined size for the eight FIFOs of each device's four channels. (This is typically half the total size for SIO8 boards.)
SIO4_FEATURE_FIFO_SIZE_TX	This is the size of the channel's Tx FIFO, or zero if the size is unknown.
SIO4_FEATURE_FORM_FACTOR	This indicates the board's basic form factor, and should be a value from the sio4_form_factor_t

	enumeration.
SIO4_FEATURE_FW_TYPE	This is a value from the sio4_fw_type_t enumeration. For Z16C30 based boards, the value returned should be SIO4_FW_TYPE_Z16C30.
SIO4_FEATURE_FW_TYPE_CONFIG	This indicates if the board supports both the SYNC firmware and the Z16C30 firmware. If it does, then the SIO4_FEATURE_FW_TYPE_CONFIG IOCTL service (section 6.4.3, page 44) can be used to select the firmware type.
SIO4_FEATURE_IRQ_32	Are all 32-bits of the interrupt configuration registers significant?
SIO4_FEATURE_FIFO_SPACE_CFG	This indicates if the firmware supports the option of configuring the amount of FIFO space for the receiver and transmitter.
SIO4_FEATURE_INDEX_BOARD	This returns the zero-based index of the board. Each SIO4 counts as one board. Each SIO8 counts as two boards.
SIO4_FEATURE_INDEX_CHANNEL	This returns the channel index on the board. Values returned are from zero to three.
SIO4_FEATURE_INDEX_DEVICE	This returns the zero-based serial channel device index relative to all serial channels.
SIO4_FEATURE_INDEX_SUBDEVICE	This is the sub-device index for the SIO4 being accessed. The value will be zero for SIO4 boards and zero or one for SIO8 boards.
SIO4_FEATURE_LED_CHANNEL	This indicates the number of LEDs on the board dedicated to each channel.
SIO4_FEATURE_LED_MAIN	This indicates the number of LEDs on the board that are not associated with the serial channels.
SIO4_FEATURE_LEGACY_CABLE	Does the firmware include the legacy cable interface control?
SIO4_FEATURE_MODEL_BASE	This returns the base model of the board and is a member of the sio4_model_t enumeration.
SIO4_FEATURE_MODEL_SYNC	Is this a SYNC based version of the SIO4? This is one for yes and zero for no. (Please note that on some boards the firmware type may be selectable (section 6.4.3, page 44).)
SIO4_FEATURE_MODEL_Z16C30	Is this a Zilog Z16C30 based version of the SIO4? This is one for yes and zero for no. (Please note that on some boards the firmware type may be selectable (section 6.4.3, page 44).)
SIO4_FEATURE_MP	Is the Multi-Protocol transceiver feature in firmware?
SIO4_FEATURE_MP_BITMAP	This is a bitmap of the transceiver options supported by the board. The bits correspond to the protocols given by the sio4_mp_prot_t (section 6.2.5, page 23) enumeration.
SIO4_FEATURE_MP_CHIP	Which Multi-Protocol transceiver chip is present?
SIO4_FEATURE_MP_PROGRAM	Can a transceiver selection be reprogrammed?
SIO4_FEATURE_OSC_CHIP	Which programmable oscillator chip is present?
SIO4_FEATURE_OSC_MEASURE	Is the driver able to measure the oscillator's frequency?
SIO4_FEATURE_OSC_PD_MAX	This is the maximum value that can be assigned to the firmware post dividers.
SIO4_FEATURE_OSC_PER_CHANNEL	Is each channel separately and individually programmable?

SIO4_FEATURE_OSC_PROGRAM	Is the driver able to program the oscillator?
SIO4_FEATURE_REG_BSR	Is the Board Status Register supported?
SIO4_FEATURE_REG_CCR	Are the Clock Control Registers supported?
SIO4_FEATURE_REG_FCR	Are the FIFO Count registers supported?
SIO4_FEATURE_REG_FR	Is the Features Register supported?
SIO4_FEATURE_REG_FSR	Are the FIFO Size registers supported?
SIO4_FEATURE_REG_FTR	Is the Firmware Type Register supported?
SIO4_FEATURE_REG_GPIOSR	Is the GPIO Source Register supported?
SIO4_FEATURE_REG_IELR	Is the Interrupt Edge/Level Register supported?
SIO4_FEATURE_REG_IHLR	Is the Interrupt High/Low Register supported?
SIO4_FEATURE_REG_IOCR	Is the I/O Control Register supported?
SIO4_FEATURE_REG_PCR	Is the Programmable Clock Register supported?
SIO4_FEATURE_REG_POCSR	Is the Programmable Oscillator Control/Status Register supported?
SIO4_FEATURE_REG_PORAR	Is the Programmable Oscillator RAM Address Register supported?
SIO4_FEATURE_REG_PORDR	Is the Programmable Oscillator RAM Data Register supported?
SIO4_FEATURE_REG_PORD2R	Is the Programmable Oscillator RAM Data Register 2 supported?
SIO4_FEATURE_REG_PSRCR	Are the Pin Source Registers supported?
SIO4_FEATURE_REG_PSRCR_BITS	This is a bitmap of supported bits in the Pin Source Register. This is zero if the register is not supported.
SIO4_FEATURE_REG_PSTSR	Are the Pin Status Registers supported?
SIO4_FEATURE_REG_PSTSR_BITS	This is a bitmap of supported bits in the Pin Status Register. This is zero if the register is not supported.
SIO4_FEATURE_REG_RCR	Is the Rx Count Register supported?
SIO4_FEATURE_REG_SBR	Is the Sync Byte Register supported?
SIO4_FEATURE_REG_TCR	Is the Tx Count Register supported?
SIO4_FEATURE_REG_TSR	Is the Timestamp Register supported?
SIO4_FEATURE_RX_FIFO_FULL_CFG	Does the Control/Status Register support the channel specific Rx FIFO Full Configuration bit?
SIO4_FEATURE_RX_FIFO_FULL_CFG_GLB	Does the Board Control Register support the global Rx FIFO Full Configuration bit?
SIO4_FEATURE_RX_FIFO_OVERRUN	Does the board support the Rx FIFO Overrun feature?
SIO4_FEATURE_RX_FIFO_UNDERRUN	Does the board support the Rx FIFO Underrun feature?
SIO4_FEATURE_RX_STATUS_WORD	Does the board support the feature of including the USC Rx Status Register in the data stream?
SIO4_FEATURE_SIO4_TYPE	This indicates the basic model type for the SIO4 and should be a value from the sio4_type_t enumeration.
SIO4_FEATURE_TIME_STAMP	Does the board support the Time stamping feature?
SIO4_FEATURE_TX_FIFO_EMPTY_CFG	Does the board support the channel specific Tx FIFO Full Configuration bit?
SIO4_FEATURE_TX_FIFO_OVERRUN	This indicates if the board supports the Tx FIFO Overrun feature.
SIO4_FEATURE_USER_JUMPER_QTY	This is the number of jumpers supported by the board.
SIO4_FEATURE_USER_JUMPER_SENSE	This is the bit value returned if a jumper is present. This is zero if no jumpers are supported.
SIO4_FEATURE_USER_JUMPER_VAL	This is the value read from the user jumpers pins. This is zero if no jumpers are supported.

The table below lists common response values for most the feature options.

Macros	Description
SIO4_FEATURE_NO	The feature is not supported.
SIO4_FEATURE_UNKNOWN	Either the feature is unknown or support for the feature is unknown.
SIO4_FEATURE_YES	The feature is supported.

6.4.3. SIO4_FW_TYPE_CONFIG

This service configures the channel for operation under the specified firmware type. If one of the predefined firmware types is requested, then it is applied. If the XXX_READ macro is supplied, then the current firmware type is not changed. Before returning, the current configuration is obtained and reported to the caller. If the feature is not configurable on the current board, then no change can be applied.

Later SIO4 models include firmware for both SYNC and Zilog based operation and allow applications to change the current firmware type on a per-channel-basis. As of driver release version 1.59, the instances where the driver changes the firmware type has been reduced. Accordingly, the driver changes the current firmware type only under the following circumstances.

1. When the driver is loaded the firmware type for all four channels is set to the board's default.

2. When the Initialize Board IOCTL service is called (section 6.4.5, page 45), the firmware type for all four channels is set to the board's default.

3. When an application calls the Firmware Type Configuration IOCTL service, the firmware type for the accessed channel is updated as requested.

> **NOTE**: It is recommended that the firmware type be changed once only, as required for application operation. It is recommended that the firmware type not me changed repeatedly.
>
> **NOTE**: Refer to the SIO4_FEATURE_FW_TYPE_CONFIG feature option to determine availability of this feature (section 6.4.2, page 41).
>
> **NOTE**: Selecting the Z16C30 firmware type results in the entire USC chip being reset, which affects both channels using that chip (i.e., channels 1 and 2 or channels 3 and 4).

Usage

ioctl() Argument	Description
request	SIO4_FW_TYPE_CONFIG
arg	__s32*

The table below lists the options used by this service.

Macros	Description
SIO4_FW_TYPE_CONFIG_READ	This is used to retrieve the current configuration.
SIO4_FW_TYPE_CONFIG_SYNC	This refers to the SYNC firmware, which is the default for all – SYNC model SIO4 boards.
SIO4_FW_TYPE_CONFIG_Z16C30	This refers to the Z16C30 firmware, which is the default for all non–SYNC model SIO4 boards. For driver support under this firmware please refer to the appropriate user manual.

6.4.4. SIO4_GET_DRIVER_INFO

This service retrieves information about the driver itself. At this time this includes only a driver version string.

Usage

ioctl() Argument	Description
request	SIO4_GET_DRIVER_INFO
arg	Sio4_driver_info_t* (section 6.2.2, page 22)

6.4.5. SIO4_INIT_BOARD

This service initializes all of the board's hardware for all four channels. This includes the transmitters, the receivers, the FIFOs, the cable configurations, the transceivers and the programmable oscillators. For boards with programmable oscillators and programmable transceivers, these features are initialized in preparation for use.

WARNING: This service affects all four channels on the board and should be used with care.

NOTE: If the firmware type is configurable, this service resets the firmware type for all four channels to the board's default.

Usage

ioctl() Argument	Description
request	SIO4_INIT_BOARD
arg	Not used.

6.4.6. SIO4_INT_NOTIFY

This service requests that the application be notified of one or more interrupts on the given serial channel. The parameter value is the bit wise or-ing of the possible notification bits. (The bits are defined in a previous section of this document.) Notification is given only for those bits which are set. Passing in a value of zero (0) cancels all notification requests. Once a specified interrupt occurs the driver clears and disables the interrupt, then notifies the application via a SIGIO (from signal.h) signal. To receive any subsequent notifications the application must make another notification request. The referenced interrupts are enabled. Unreferenced interrupts are disabled.

NOTE: Interrupt options referenced but unsupported by the current hardware are quietly ignored.

Usage

ioctl() Argument	Description
request	SIO4_INT_NOTIFY
arg	unsigned char

The table below lists the options used with this service. These options may be used in any bitwise combination.

Macros	Description
SIO4_INT_NOTIFY_RX_FIFO_AF	The Rx FIFO Almost Full interrupt.
SIO4_INT_NOTIFY_RX_FIFO_E	The Rx FIFO Empty interrupt.
SIO4_INT_NOTIFY_RX_FIFO_F	The Rx FIFO Full interrupt.
SIO4_INT_NOTIFY_TX_FIFO_AE	The Tx FIFO Almost Empty interrupt.
SIO4_INT_NOTIFY_TX_FIFO_E	The Tx FIFO Empty interrupt.
SIO4_INT_NOTIFY_TX_FIFO_F	The Tx FIFO Full interrupt.

6.4.7. SIO4_MOD_REGISTER

This service performs a read-modify-write operation on an SIO4 register. This includes only the GSC firmware registers and USC registers. The PCI registers and the PLX feature set registers are read-only. Refer to the SIO4 User Manual and to `sio4.h` for a complete list of the available registers.

Usage

ioctl() Argument	**Description**
`request`	`SIO4_MOD_REGISTER`
`arg`	`Sio4_reg_t*` (section 6.2.9, page 25)

6.4.8. SIO4_MP_CONFIG

This service is used to select and/or report on the current transceiver protocol. The driver uses the `prot_want` field and ignores all others. The results are recorded in the data structure's `prot_got` field. Refer to the Multi-Protocol transceiver programming information later in this document for more information.

> **NOTE:** The driver will fulfill the request based on the SIO4's capabilities. When the protocol can be changed and that requested is available, the requested change will be selected. Requests will otherwise fail and the protocol will be unchanged.

Usage

ioctl() Argument	**Description**
`request`	`SIO4_MP_CONFIG`
`arg`	`sio4_mp_t*` (section 6.2.6, page 23)

6.4.9. SIO4_MP_INFO

This service returns information about the current Multi-Protocol transceiver configuration. All field contents are ignored and are set by the driver according to the current configuration. Refer to the Multi-Protocol transceiver programming information later in this document for more information.

Usage

ioctl() Argument	**Description**
`request`	`SIO4_MP_INFO`
`arg`	`sio4_mp_t*` (section 6.2.6, page 23)

6.4.10. SIO4_MP_INIT

This service initializes the board's Multi-Protocol transceiver feature. This returns the Multi-Protocol transceivers to their initial power up state. The results are recorded in the data structure's `prot_got` field. Refer to the Multi-Protocol transceiver programming information later in this document for more information.

Usage

ioctl() Argument	**Description**
`request`	`SIO4_MP_INIT`
`arg`	`sio4_mp_t*` (section 6.2.6, page 23)

6.4.11. SIO4_MP_RESET

This service resets the board's Multi-Protocol transceiver feature. This disables the transceivers by tri-stating the outputs. The results are recorded in the data structure's `prot_got` field. Refer to the Multi-Protocol transceiver programming information later in this document for more information.

Usage

ioctl() Argument	**Description**
request	SIO4_MP_RESET
arg	sio4_mp_t* (section 6.2.6, page 23)

6.4.12. SIO4_MP_TEST

This service is used to determine if the board's Multi-Protocol transceiver feature supports a given protocol. The protocol to be tested is recorded in the structure's `prot_want` field. The results are recorded in the data structure's `prot_got` field. The reported value will be `SIO4_MP_PROT_INVALID` if the requested protocol value is unrecognized or unsupported. It will be `SIO4_MP_PROT_UNKNOWN` when support for the specified protocol is unknown. This is applicable when the SIO4 doesn't support the feature or when the chip used is unsupported by the driver. The reported value will equal the requested protocol when that protocol is supported. Refer to the Multi-Protocol transceiver programming information later in this document for more information.

Usage

ioctl() Argument	**Description**
request	SIO4_MP_TEST
arg	sio4_mp_t* (section 6.2.6, page 23)

6.4.13. SIO4_OSC_INFO

This service returns current configuration information about the channel's oscillator. The driver ignores the structure's current content and fills in all fields according to the channel's current configuration. Refer to the oscillator programming information later in this document for more information.

Usage

ioctl() Argument	**Description**
request	SIO4_OSC_INFO
arg	sio4_osc_t* (section 6.2.8, page 24)

6.4.14. SIO4_OSC_INIT

This service initializes the channel's programmable oscillator hardware. The channel's input clock will be reprogrammed to output the reference frequency as a result of this service, depending on the device's capabilities. The driver ignores the structure's current content and fills in all fields according to the channel's post-initialization configuration. The reference frequency is unaltered, the desired frequency is set to the reference frequency, and the frequency obtained is reported. Refer to the oscillator programming information later in this document for more information.

Usage

ioctl() Argument	**Description**
request	SIO4_OSC_INIT
arg	sio4_osc_t* (section 6.2.8, page 24)

6.4.15. SIO4_OSC_MEASURE

This service is used to measure the frequency produced by the current oscillator hardware configuration. The driver ignores all structure field values and fills them in according to the test results and the channel's current configuration. The test results are recorded in the data structure's `freq_got` field. A value of `-1` is reported when the frequency can't be measured. Refer to the oscillator programming information later in this document for more information.

> **NOTE:** The driver will perform a measurement test based on the SIO4's capabilities. When a measurement can be made, the test duration and the accuracy of the results are dependent on the board's capabilities. Refer to the hardware manual for additional details.

Usage

ioctl() Argument	**Description**
`request`	`SIO4_OSC_MEASURE`
`arg`	`sio4_osc_t*` (section 6.2.8, page 24)

6.4.16. SIO4_OSC_PROGRAM

This service is used to update and report on the programmed frequency produced by the channel's oscillator hardware. This service will reprogram the channel's oscillator hardware to produce the requested frequency, or one as near as possible to that requested. The resulting frequency will depend on the capability of the hardware and how its resources are being used, as applicable. If the requested value is `-1`, then the service will report the channel's current configuration without making any changes. The driver ignores all other fields and fills them in according to the channel's post-programming configuration. Refer to the oscillator programming information later in this document for more information.

Usage

ioctl() Argument	**Description**
`request`	`SIO4_OSC_PROGRAM`
`arg`	`sio4_osc_t*` (section 6.2.8, page 24)

6.4.17. SIO4_OSC_REFERENCE

This service is used to update and report on the recorded frequency for the channel's reference source. Changing this setting does not alter any existing programming results. New settings apply to subsequent calculations only! The only argument field used by the driver is the `freq_ref` field. If its value is `-1`, then the driver will report the current recorded reference frequency. The value supplied will otherwise be qualified per the requirements of the channel's oscillator and recorded for subsequent use. An error will be reported if it is invalid. The driver ignores all other fields and fills them in according to the channel's current configuration. This service does not alter any other oscillator related parameter. Refer to the oscillator programming information later in this document for more information.

> **CAUTION:** Setting the reference frequency to an incorrect value may have an adverse effect on the programmable oscillator. The results depend on the oscillator and the incorrect value specified.

Usage

ioctl() Argument	**Description**
`request`	`SIO4_OSC_REFERENCE`
`arg`	`sio4_osc_t*` (section 6.2.8, page 24)

6.4.18. SIO4_OSC_RESET

This service resets the channel's oscillator hardware. The channel's input clock will be set to the lowest possible frequency as a result of this service, depending on the device's capabilities. The driver ignores the structure's current content and fills in all fields according to the channel's post-reset configuration. The reference frequency is unaltered, the desired frequency is set to zero, and the frequency obtained is reported. Refer to the oscillator programming information later in this document for more information.

Usage

ioctl() Argument	**Description**
request	SIO4_OSC_RESET
arg	sio4_osc_t* (section 6.2.8, page 24)

6.4.19. SIO4_OSC_TEST

This service reports the frequency that should be produced were the programming service requested for the desired frequency. The channel's input clock will be set to the lowest possible frequency as a result of this service, depending on the device's capabilities. The driver ignores the structure's current content and fills in all fields according to the channel's post-reset configuration. The reference frequency is unaltered, the desired frequency is set to zero, and the frequency obtained is reported. Refer to the oscillator programming information later in this document for more information.

Usage

ioctl() Argument	**Description**
request	SIO4_OSC_TEST
arg	sio4_osc_t* (section 6.2.8, page 24)

6.4.20. SIO4_READ_INT_STATUS

This service requests the interrupt status information following interrupt notification. The status reported reflects all of the interrupts for the channel. The recorded status represents the accumulated status of all interrupts since the status was last read or notification requested. Once read, the recorded status is cleared.

> **NOTE:** Due to the timeliness of various interacting events it is possible for multiple interrupts to occur before the status is read. This can result in one SIGIO prompted status read reporting multiple interrupts and the next SIGIO prompted status read reporting no interrupts.

Usage

ioctl() Argument	**Description**
request	SIO4_READ_INT_STATUS
arg	SIO4_INTERRUPT_STATUS* (section 6.2.3, page 22)

6.4.21. SIO4_READ_REGISTER

This service reads the value of an SIO4 register. This includes all PCI registers, all PLX feature set registers, and all GSC firmware registers for the referenced channel. Refer to the SIO4 User Manual and to sio4.h for a complete list of the available registers.

Usage

ioctl() Argument	Description
request	SIO4_READ_REGISTER
arg	Sio4_reg_t* (section 6.2.9, page 25)

6.4.22. SIO4_READ_REGISTER_RAW

This service reads the value of an SIO4 firmware register without respect to the channel being accessed. This applies to firmware registers only. Permissible values are from zero to 63. All other values result in failure. Refer to the SIO4 User Manual and to `sio4.h` for a complete list of the predefined register identifiers.

Usage

ioctl() Argument	Description
request	SIO4_READ_REGISTER_RAW
arg	Sio4_reg_t* (section 6.2.9, page 25)

6.4.23. SIO4_RESET_CHANNEL

This service performs a reset of the entire channel. This includes the transmitter, the receiver, the FIFOs, the cable configuration, the transceivers and the programmable oscillator. (The programmable oscillator is reset only if the SIO4 supports a different programmable source for each channel.)

> **NOTE**: If the firmware type is configurable, it is left unchanged. Thus, only those resources for the current Firmware Type are reset.

Usage

ioctl() Argument	Description
request	SIO4_RESET_CHANNEL
arg	Not used.

6.4.24. SIO4_RESET_DEVICE

This service resets all of the board's hardware. This includes the transmitter, the receiver, the FIFOs, the cable configurations, the transceivers and the programmable oscillators. The programmable transceivers and programmable oscillators are disabled, if supported in hardware.

> **WARNING:** This service affects all four channels on the board and should be used with care.

> **NOTE**: If the firmware type is configurable, this service resets the firmware type for all four channels to the board's default.

Usage

ioctl() Argument	Description
request	SIO4_RESET_DEVICE
arg	Not used.

6.4.25. SIO4_RESET_FIFO

This service resets either or both of the channel FIFOs.

Usage

ioctl() Argument	Description
request	SIO4_RESET_FIFO
arg	TX_RX* (section 6.2.13, page 28)

6.4.26. SIO4_RX_FIFO_AE_CONFIG

This service configures the Rx FIFO Almost Empty level and reports the current level. When applying a setting, the Rx FIFO is reset and the current content is lost. If the XXX_READ macro is supplied then no change is applied. Before returning the current programmed level is obtained and supplied to the caller.

Usage

ioctl() Argument	Description
request	SIO4_RX_FIFO_AE_CONFIG
arg	__s32*

6.4.27. SIO4_RX_FIFO_AF_CONFIG

This service configures the Rx FIFO Almost Full level and reports the current level. When applying a setting, the Rx FIFO is reset and the current content is lost. If the XXX_READ macro is supplied then no change is applied. Before returning the current programmed level is obtained and supplied to the caller.

Usage

ioctl() Argument	Description
request	SIO4_RX_FIFO_AF_CONFIG
arg	__s32*

6.4.28. SIO4_RX_FIFO_COUNT

This service retrieves the current Rx FIFO fill level. The value obtained is either the number of bytes of data in the Rx FIFO or the XXX_UNKNOWN macro if the Rx FIFO Count Register is unsupported.

Usage

ioctl() Argument	Description
request	SIO4_RX_FIFO_COUNT
arg	__s32*

The value returned is from zero to the size of the FIFO or the value given below.

Macros	Description
SIO4_FIFO_COUNT_UNKNOWN	The FIFO fill level is unknown.

6.4.29. SIO4_RX_FIFO_FULL_CFG_CHAN

This service configures the channel specific setting for how the receiver responds to an Rx FIFO Full condition and reports on the current configuration. If one of the predefined configurations is requested, it is applied. If the XXX_READ macro is supplied, then the current configuration is not changed. Before returning, the current configuration is obtained and reported to the caller. If the feature is not configurable on the current board, then no change can be applied. The channel specific setting is ignored if the global setting is the *over* option.

Usage

ioctl() Argument	Description
request	SIO4_RX_FIFO_FULL_CFG_CHAN
arg	__s32*

The table below lists the options used by this service.

Macros	Description
SIO4_RX_FIFO_FULL_CFG_CHAN_READ	This is used to retrieve the current configuration.
SIO4_RX_FIFO_FULL_CFG_CHAN_HALT	Disable the FIFO and halt the inflow of data.
SIO4_RX_FIFO_FULL_CFG_CHAN_OVER	Let the FIFO overrun by discarding excess data.

6.4.30. SIO4_RX_FIFO_FULL_CFG_GLB

This service configures the global setting for how the receivers respond to an Rx FIFO Full condition and reports on the current configuration. If one of the predefined configurations is requested, it is applied. If the XXX_READ macro is supplied, then the current configuration is not changed. Before returning, the current configuration is obtained and reported to the caller. If the feature is not configurable on the current board, then no change can be applied.

Usage

ioctl() Argument	Description
request	SIO4_RX_FIFO_FULL_CFG_GLB
arg	__s32*

The table below lists the options used by this service.

Macros	Description
SIO4_RX_FIFO_FULL_CFG_GLB_READ	This is used to retrieve the current configuration.
SIO4_RX_FIFO_FULL_CFG_GLB_HALT	Disable the receiver and halt the inflow of data. This setting overrides the per channel settings, if supported.
SIO4_RX_FIFO_FULL_CFG_GLB_OVER	Let the FIFO overrun by discarding excess data. With this setting, the per channel setting take effect, if supported.

6.4.31. SIO4_RX_FIFO_SIZE

This service retrieves the size of the Rx FIFO. The value obtained is either the capacity of the Rx FIFO in bytes or zero if the size is unknown.

Usage

ioctl() Argument	Description
request	SIO4_RX_FIFO_SIZE
arg	__s32*

6.4.32. SIO4_RX_FIFO_STATUS

This service retrieves the Rx FIFO fill level status. The value obtained reflects the FIFO's relative fill level.

Usage

ioctl() Argument	Description
request	SIO4_RX_FIFO_SIZE
arg	__s32*

The value returned should be one of the below listed options.

Value	Description
SIO4_FIFO_STATUS_EMPTY	The FIFO is empty.
SIO4_FIFO_STATUS_ALMOST_EMPTY	The FIFO contains *Almost Empty* or fewer data values (section 6.4.26, page 51).
SIO4_FIFO_STATUS_MEDIAN	The FIFO fill level is between the Almost Empty and Almost Full levels.
SIO4_FIFO_STATUS_ALMOST_FULL	The FIFO can receive *Almost Full* or fewer data value before becoming full (section 6.4.27, page 51).
SIO4_FIFO_STATUS_FULL	The FIFO is full.

6.4.33. SIO4_RX_IO_ABORT

This service aborts a `read()` operation. This service waits for up to 10 seconds to abort either a currently active `read()` operation or one that is initiated during the abort waiting period.

Usage

ioctl() Argument	Description
request	SIO4_RX_IO_ABORT
arg	__s32*

The table below lists the options used with this service.

Macros	Description
0	An abort did not take place.
1	An abort did take place.

6.4.34. SIO4_RX_IO_MODE_CONFIG

This service updates and reports the mode used by the driver for data read operations. This refers to how data is moved from the SIO4 to host memory when the `read()` function is called.

Usage

ioctl() Argument	Description
request	SIO4_RX_IO_MODE_CONFIG
arg	__s32*

The table below lists the options used with this service.

Macros	Description
SIO4_IO_MODE_DEFAULT	This refers to the default I/O mode, which is PIO.
SIO4_IO_MODE_BMDMA	This refers to Block Mode DMA, which is generally performed without regard to the FIFO's content.
SIO4_IO_MODE_DMDMA	This refers to Demand Mode DMA, which transfers data as it becomes available.

SIO4_IO_MODE_PIO	This refers to PIO, which uses repetitive register accesses.
SIO4_IO_MODE_READ	This is used to retrieve the current configuration.

6.4.35. SIO4_SET_READ_TIMEOUT

This service sets the timeout limit for read requests, and is the maximum amount of time the driver will wait for a blocking `read()` request to complete. The timeout period is specified in seconds. Timeout values of zero (0) or less mean do not wait.

Usage

ioctl() Argument	**Description**
request	SIO4_SET_READ_TIMEOUT
arg	__u32

6.4.36. SIO4_SET_WRITE_TIMEOUT

This service sets the timeout limit for write requests, and is the maximum amount of time the driver will wait for a blocking `write()` request to complete. The timeout period is specified in seconds. Timeout values of zero (0) or less mean do not wait.

Usage

ioctl() Argument	**Description**
request	SIO4_SET_WRITE_TIMEOUT
arg	__u32

6.4.37. SIO4_TX_FIFO_AE_CONFIG

This service configures the Tx FIFO Almost Empty level and reports the current level. When applying a setting, the Tx FIFO is reset and the current content is lost. If the `XXX_READ` macro is supplied then no change is applied. Before returning the current programmed level is obtained and supplied to the caller.

Usage

ioctl() Argument	**Description**
request	SIO4_TX_FIFO_AE_CONFIG
arg	__s32*

6.4.38. SIO4_TX_FIFO_AF_CONFIG

This service configures the Tx FIFO Almost Full level and reports the current level. When applying a setting, the Tx FIFO is reset and the current content is lost. If the `XXX_READ` macro is supplied then no change is applied. Before returning the current programmed level is obtained and supplied to the caller.

Usage

ioctl() Argument	**Description**
request	SIO4_TX_FIFO_AF_CONFIG
arg	__s32*

6.4.39. SIO4_TX_FIFO_COUNT

This service retrieves the current Tx FIFO fill level. The value obtained is either the number of bytes of data in the Tx FIFO or the `XXX_UNKNOWN` macro if the Tx FIFO Count Register is unsupported.

Usage

ioctl() Argument	**Description**
request	SIO4_TX_FIFO_COUNT
arg	__s32*

The value returned is from zero to the size of the FIFO or the value given below.

Macros	**Description**
SIO4_FIFO_COUNT_UNKNOWN	The FIFO fill level is unknown.

6.4.40. SIO4_TX_FIFO_SIZE

This service retrieves the size of the Tx FIFO. The value obtained is either the capacity of the Tx FIFO in bytes or zero if the size is unknown.

Usage

ioctl() Argument	**Description**
request	SIO4_TX_FIFO_SIZE
arg	__s32*

6.4.41. SIO4_TX_FIFO_STATUS

This service retrieves the Tx FIFO fill level status. The value obtained reflects the FIFO's relative fill level.

Usage

ioctl() Argument	**Description**
request	SIO4_TX_FIFO_SIZE
arg	__s32*

The value returned should be one of the below listed options.

Value	**Description**
SIO4_FIFO_STATUS_EMPTY	The FIFO is empty.
SIO4_FIFO_STATUS_ALMOST_EMPTY	The FIFO contains *Almost Empty* or fewer data values (section 6.4.37, page 54).
SIO4_FIFO_STATUS_MEDIAN	The FIFO fill level is between the Almost Empty and Almost Full levels.
SIO4_FIFO_STATUS_ALMOST_FULL	The FIFO can receive *Almost Full* or fewer data value before becoming full (section 6.4.38, page 54).
SIO4_FIFO_STATUS_FULL	The FIFO is full.

6.4.42. SIO4_TX_IO_ABORT

This service aborts a `write()` operation. This service waits for up to 10 seconds to abort either a currently active `write()` operation or one that is initiated during the abort waiting period.

Usage

ioctl() Argument	Description
request	SIO4_TX_IO_ABORT
arg	__s32*

The table below lists the options used with this service.

Macros	Description
0	An abort did not take place.
1	An abort did take place.

6.4.43. SIO4_TX_IO_MODE_CONFIG

This service updates and reports the mode used by the driver for data write operations. This refers to how data is moved from host memory to the SIO4 when the `write()` function is called.

Usage

ioctl() Argument	Description
request	SIO4_TX_IO_MODE_CONFIG
arg	__s32*

The table below lists the options used with this service.

Macros	Description
SIO4_IO_MODE_DEFAULT	This refers to the default I/O mode, which is PIO.
SIO4_IO_MODE_BMDMA	This refers to Block Mode DMA, which is generally performed without regard to the FIFO's content.
SIO4_IO_MODE_DMDMA	This refers to Demand Mode DMA, which transfers data as space becomes available.
SIO4_IO_MODE_PIO	This refers to PIO, which uses repetitive register accesses.
SIO4_IO_MODE_READ	This is used to retrieve the current configuration.

6.4.44. SIO4_WRITE_REGISTER

This service writes a value to an SIO4 register. This includes GSC firmware registers and the USC registers only. All PCI and PLX feature set registers are read-only. Refer to the SIO4 User Manual and to `sio4.h` for a complete list of available registers. Applications should exercise care in writing to some of these registers. This is because some are used by the driver for interrupt and DMA purposes. Writing to these registers may interfere with proper SIO4 and driver operation and may disrupt the stability of the operating system. The registers of concern are those listed below.

- The GSC Board Control Register
- The GSC Interrupt Control Register (and the interrupt configuration registers)
- The GSC Interrupt Status Register

Usage

ioctl() Argument	Description
request	SIO4_WRITE_REGISTER
arg	Sio4_reg_t* (section 6.2.9, page 25)

7. Operation

This section explains some operational procedures on using the driver. This is in no way intended to be a comprehensive guide on using the SIO4. This is simply to address a very few issues relating to GSC specific features of the SIO4.

7.1. I/O Modes

The following describes the three supported I/O modes used for data transfer between the host and the SIO4. All three modes are available using the C library routines `read()` and `write()`. Applications select the desired mode using IOCTL services. Use the `SIO4_TX_IO_MODE_CONFIG` IOCTL service to configure the `write()` data transfer mode and use the `SIO4_RX_IO_MODE_CONFIG` IOCTL service to configure the `read()` data transfer mode.

7.1.1. PIO - Programmed I/O

This mode uses repetitive register accesses. While it is the least efficient method it accommodates simultaneous transfers on any number of channels and in both directions. Applications can make PIO mode I/O requests without having to monitor FIFO fill levels.

7.1.2. BMDMA - Block Mode DMA

This refers to Block Mode DMA. This mode transfers data with little CPU overhead, but is suitable only for requests that do not exceed the size of the installed FIFOs. Using this mode, applications much monitor a FIFO's fill level to ensure that it can accommodate desired requests. Calling `read()` when the Rx FIFO contains insufficient data will result in indeterminate data at the point where the FIFO runs empty. Calling `write()` when the Tx FIFO contains insufficient free space will result in data loss at the point the FIFO becomes full. Since the SIO4 can have up to eight data streams (4 Rx and 4 Rx) and only two DMA engines are available, applications must make selective use of DMA and non-DMA I/O requests.

7.1.3. DMDMA - Demand Mode DMA

This mode transfers data with the least amount of CPU overhead. It accommodates transfers that exceed the size of the installed FIFOs and uses the FIFO fill level to throttle data movement over the PCI bus. This permits efficient data movement over the PCI bus and also permits the transfer to remain active while data is being transferred over the cable interface. Since the SIO4 can have up to eight data streams (4 Rx and 4 Rx) and only two DMA engines are available, applications must make selective use of DMA and non-DMA I/O requests. Applications can make DMDMA mode I/O requests without having to monitor FIFO fill levels.

7.2. Oscillator Programming

The ability to program the SIO4's onboard oscillators depend on the board's hardware capabilities and on support included in the driver. The driver can identify the oscillator chip for all SIO4 implementations up to and including those using the Cypress CY22393 Programmable Oscillator. At present however, the driver includes built-in programming support only for those SIO4s using a single CY22393. The driver will return an error status when exercising the programmable oscillator features for all other programmable oscillator types. The general procedure to follow when using the programmable oscillator features are as follows.

> **NOTE:** The driver measures the SIO4's reference frequency when the driver is first loaded. If it cannot be measures, then it is initialized to 20MHz. Thereafter, the reference frequency is changed only when done explicitly by application requests using the `SIO4_OSC_REFERENCE` IOCTL service.

1. Determine if the driver is able to perform oscillator programming for the device. This can be done using the SIO4_FEATURE_TEST IOCTL service on the SIO4_FEATURE_OSC_PROGRAM feature. If the feature in unsupported, then do not attempt programming. Attempting to use the driver's built-in programming features will be unsuccessful when this feature is unsupported. If the feature is supported, then continue with the following steps.

2. Tell the driver the SIO4's reference frequency. This is done using the SIO4_OSC_REFERENCE IOCTL service. The specified reference frequency is applicable to all channels since the SIO4 has only a single reference oscillator. The specified reference frequency is used for subsequent operations only.

3. Reset the channel's clock. This is done using the SIO4_OSC_RESET IOCTL service. Depending on the oscillator, this may disable the channel's clock. Depending on the SIO4, this effort may affect all channels.

4. Initialize the channel's clock. This is done using the SIO4_OSC_INIT IOCTL service. Depending on the oscillator, this should configure the channel to output the reference frequency. Depending on the SIO4, this effort may affect all channels.

5. Request that the oscillator be reprogrammed for the desired frequency. This is done using the SIO4_OSC_PROGRAM IOCTL service. The resulting frequency will be as close as possible to the requested frequency. How close this actually is depends on the oscillator's capabilities, its current resource usage and the reference frequency. Check the sio4_osc_t (section 6.2.8, page 24) structure's freq_got field after programming to verify that the resulting frequency is sufficient. Depending on the SIO4, the programming effort may affect all channels.

   NOTE: On occasion, the oscillator programming effort may not take full affect even though the operation completes successfully. Applications should therefore measure the oscillator frequency following programming requests. If the measured results differ significantly from what the programming request indicated would be produced, then repeat the programming and measurement steps until the results are satisfactory.

6. If desired, the channel's current frequency can be measured at any time using the SIO4_OSC_MEASURE IOCTL service. However, this should only be done if the frequency can be measured. This capability depends on the SIO4's feature set. Support for this feature can be determined by using the SIO4_FEATURE_TEST IOCTL service with the SIO4_FEATURE_OSC_MEASURE feature argument.

7. If desired, the current configuration may be determined at any time using the SIO4_OSC_INFO IOCTL service. The information returned will be based on the driver's recorded state information.

7.2.1. Cypress CY22393 (1x) Programmable Oscillator Support

The SIO4's support for this device includes a fixed reference oscillator, a Cypress CY22393 (with four programmable oscillators), and four firmware-based post dividers. The driver defaults the reference frequency to the measured frequency at startup and initializes the programmable oscillators to their off state. The driver manages the firmware post dividers and the CY22393, with its oscillators and Digital Phase Lock Loop Generators, as best as possible to fulfill application requests. When a programming request is made the driver applies the appropriate changes, measures the results, and reprograms the changes as necessary. The measurement and reprogramming steps occur when a channel is opened and closed, and when operations are requested by an application. The driver responds to the services according to the following table.

Service	Response
SIO4_OSC_INFO	The current settings are reported.
SIO4_OSC_INIT	The desired frequency is set to the reference frequency and the channel is reconfigured accordingly.

SIO4_OSC_MEASURE	The output frequency is measured using SIO4 firmware resources. The measured value is reported in the freq_got field.
SIO4_OSC_PROGRAM	If the requested frequency is non-negative and 20MHz or less, then the driver programs in that configuration that will most closely match the request. This is done based on the CY22393's resources available at that moment.
SIO4_OSC_REFERENCE	The requested value is recorded if it is 8MHz or higher and 30MHz or lower.
SIO4_OSC_RESET	The desired frequency is set to zero and the channel is reconfigured accordingly.

7.2.2. Cypress CY22393 (4x) Programmable Oscillator Support

The driver does not include support for this device configuration. The driver returns EIO for all programmable oscillator requests when the SIO4 uses this chip configuration.

7.2.3. Cypress IDC2053B Programmable Oscillator Support

The driver does not include support for this device. The driver returns EIO for all programmable oscillator requests when the SIO4 uses this chip.

7.2.4. Fixed Oscillator Support

When the SIO4 has a fixed oscillator, no programming can be performed. Rather than return errors though, the driver treats the hardware as a programmable oscillator capable only of supply the reference frequency. The driver responds to the IOCTL services according to the following table.

Service	Response
SIO4_OSC_INFO	The current settings are reported.
SIO4_OSC_INIT	The freq_got value is updated to the reference frequency.
SIO4_OSC_MEASURE	The freq_got value is reported as -1 (due to firmware limitations).
SIO4_OSC_PROGRAM	The requested value is recorded if it is non-zero and 20MHz or lower.
SIO4_OSC_REFERENCE	The requested value is recorded if it is 1MHz or higher and 20MHz or lower.
SIO4_OSC_RESET	The freq_got value is updated to the reference frequency.

7.2.5. All Other Cases

This applies when the SIO4 includes no programmable oscillator support and when the SIO4 uses a programmable oscillator unrecognized by the driver. The driver responds to the IOCTL services according to the following table.

Service	Response
SIO4_OSC_INFO	The current recorded settings are reported.
SIO4_OSC_INIT	The recorded freq_want and freq_got values are set to the reference frequency.
SIO4_OSC_MEASURE	The freq_got value is reported as zero.
SIO4_OSC_PROGRAM	The recorded freq_want and freq_got values are set to the requested value if it is non-zero and 20MHz or lower.
SIO4_OSC_REFERENCE	The requested value is recorded if it is 1MHz or higher and 20MHz or lower.
SIO4_OSC_RESET	The recorded freq_want and freq_got values are set to zero.

7.3. Multi-Protocol Transceiver Programming

This feature includes boards with varying capabilities. Some boards are able to change the transceiver protocol under software control. Some have fixed transceiver protocols and can report the protocol via firmware. Others have fixed transceiver protocols, but are not able to report the protocol. The general procedure to follow when using this feature is as follows.

1. Determine if the SIO4 supports this feature. This can be done using the SIO4_FEATURE_TEST IOCTL service on the SIO4_FEATURE_MP feature. If this feature in unsupported, then do not attempt to exercise the board's Multi-Protocol transceiver feature. Attempting to do so will be unsuccessful when this feature is unsupported. If the feature is supported, then continue with the following steps.

2. Determine if the SIO4's transceiver protocol can be changed. This can be done using the SIO4_FEATURE_TEST IOCTL service on the SIO4_FEATURE_MP_CHANGE feature. If this feature is unsupported, then do not attempt to exercise the board's Multi-Protocol transceiver feature. Attempting to do so will be unsuccessful when this feature is unsupported. If the feature is supported, then continue with the following steps.

3. Determine if the transceiver protocol desired is supported. This can be done using the SIO4_MP_TEST IOCTL. If a suitable protocol cannot be selected, then do not attempt to further exercise the board's Multi-Protocol transceiver feature. If a suitable protocol is available, then continue with the following steps.

4. Select a suitable transceiver protocol. This can be done using the SIO4_MP_CONFIG IOCTL.

5. If desired, the current configuration can be determined at any time using the SIO4_OSC_INFO IOCTL service.

7.3.1. Sipex SP508 Multi-Protocol Transceiver Support

When the SIO4 includes these transceiver chips, the driver responds to the services according to the following table.

Service	Response
SIO4_MP_CONFIG	The chip will be given as the SP508 option. The resulting protocol will equal the requested protocol if it is supported. The resulting protocol will otherwise be the invalid option.
SIO4_MP_INFO	The chip will be given as the SP508 option. The desired protocol will be the read option. The resulting protocol will reflect the board's current configuration.
SIO4_MP_INIT	The chip will be given as the SP508 option. The desired and resulting protocol will both be the RS-422/485 option.
SIO4_MP_RESET	The chip will be given as the SP508 option. The desired and resulting protocol will both be the disable option.
SIO4_MP_TEST	The chip will be given as the SP508 option. The resulting protocol will be the requested protocol if it is supported. The resulting protocol will otherwise be the invalid option.

7.3.2. Fixed Protocol Support

Some SIO4s include Multi-Protocol support in firmware but not in hardware. This applies when the SIO4 has fixed transceivers whose type is reported by firmware. Under these circumstances the driver responds to the IOCTL services according to the following table.

Service	Response
SIO4_MP_CONFIG	The chip will be given as the fixed option. The resulting protocol will reflect the board's hardwired protocol.
SIO4_MP_INFO	The chip will be given as the fixed option. The desired protocol will be the read option and the resulting protocol will reflect the board's hardwired protocol.
SIO4_MP_INIT	The chip will be given as the fixed option. The desired and resulting protocols will reflect the board's hardwired protocol option.
SIO4_MP_RESET	The chip will be given as the fixed option. The desired and resulting protocols will reflect the board's hardwired protocol.
SIO4_MP_TEST	The chip will be given as the fixed option. The resulting protocol will be the test protocol if it is the board's hardwired protocol. The resulting protocol will otherwise be the invalid option.

7.3.3. All Other Cases

This applies when the firmware includes no Multi-Transceiver protocol support and when support is present but the protocol is fixed. In these cases, the driver responds to the IOCTL services according to the following table.

Service	Response
SIO4_MP_CONFIG	The chip and resulting protocol will each be given as their respective unknown options.
SIO4_MP_INFO	The desired protocol will be the read option. The chip and resulting protocol will each be given as their respective unknown options.
SIO4_MP_INIT	The chip, the desired protocol and resulting protocol will all be given as their respective unknown options.
SIO4_MP_RESET	The desired protocol will be the disable option. The chip and resulting protocol will each be given as their respective unknown options.
SIO4_MP_TEST	The chip and resulting protocol will each be given as their respective unknown options.

7.4. Interrupt Notification

Applications can make indirect use of SIO4 interrupts by using the Interrupt Notification IOCTL services. This requires the following basic steps. These steps are illustrated in the source code sample that follows.

1. Use the `fcntl` interface to register the application's signal handler.

2. Issue the `SIO4_INT_NOTIFY` IOCTL service to request notification.

3. When the SIGIO signal is received, issue the `SIO4_READ_INT_STATUS` IOCTL service to determine which interrupt occurred.

4. Perform any application required actions.

5. If additional notification is required for an interrupt that was reported then repeat steps two through five as required.

6. When finished issue the `SIO4_INT_NOTIFY` IOCTL service with an argument value of zero (0) to specify that notification be terminated.

Example

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

#include "sio4_dsl.h"

static int _fd;

static void handle_sigio(int signo)
{
    SIO4_INTERRUPT_STATUS  int_stat;
    int                    status;

    status  = ioctl(_fd, SIO4_READ_INT_STATUS, &int_stat);
```

```
    if (status == -1)
    {
        // The request failed.
    }
    else if (int_stat.u8SIO4Status & SIO4_INT_NOTIFY_TX_FIFO_AE)
    {
        // Handle the Tx FIFO Almost Empty condition.
    }
}

int sio4_async_setup(int fd)
{
    int             flags;
    unsigned char   notify;
    pid_t           pid;
    int             status;

    ioctl(fd, SIO4_INT_NOTIFY, 0);
    _fd = fd;
    signal(SIGIO, handle_sigio);
    pid = getpid();
    fcntl(fd, F_SETOWN, pid);
    flags   = fcntl(fd, F_GETFL);
    flags   |= FASYNC;
    fcntl(fd, F_SETFL, flags);
    notify  = SIO4_INT_NOTIFY_TX_FIFO_AE;
    status  = ioctl(fd, SIO4_INT_NOTIFY, notify);
    return(status);
}
```

8. 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.

8.1. Files

The library files are summarized in the tables 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/sio4_dsl.h`	This is the primary utility header file.
`lib/sio4_dsl.a`	This is the statically linkable library file.

File	Description
`sync/docsrc/*.c`	These are the C source files.
`sync/docsrc/makefile`	This is the library make file.
`sync/docsrc/makefile.dep`	This is an automatically generated make dependency file.
`include/sio4_sync_dsl.h`	This is the SYNC specific primary utility header file.
`lib/sio4_sync_dsl.a`	This is the SYNC specific statically linkable library file.

8.2. Build (Generic)

The library is built via the Overall Make Script (section 2.8, page 12), but can be built separately following the below steps.

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

8.3. Build (SYNC)

The library is built via the Overall Make Script (section 2.8, page 12), but can be built separately following the below steps.

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

8.4. Library Use

The libraries are 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 respective interface. At link time include the below listed library files with the objects being linked with the application.

File	Location
sio4_dsl.h	…/sio4/include
sio4_dsl.a	…/sio4/lib
sio4_sync_dsl.h	…/sio4/include
sio4_sync_dsl.a	…/sio4/lib

9. Utility Source Code

The driver archive includes bodies of utility services built into statically linkable libraries that are 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.

9.1. 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/*.h	These are local header files.
utils/makefile	This is the library make file.
utils/makefile.dep	This is an automatically generated make dependency file.
include/sio4_utils.h	This is the primary utility header file.
lib/sio4_utils.a	This is the statically linkable library file.

9.2. Build

The libraries are built via the Overall Make Script (section 2.8, page 12), but can be built separately following the below steps.

1. Change to the directory where the utility sources are installed (see above table).

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
   ```

9.3. 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
sio4_utils.h	…/sio4/include/
sio4_utils.a	…/sio4/lib/

10. 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 2.8, page 12), 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.

> **NOTE:** These sample applications are designed to function with the SIO4 models listed on the cover of this user manual. The sample applications may work with other models, but may not function as expected since they are not necessarily intended for those models. Refer to the driver user manual and sample applications supplied with the SIO4 model in question, as applicable.
>
> **NOTE**: None of the sample application are specifically written to support simultaneous execution. The applications may function satisfactorily when multiple instances are run simultaneously on the same serial channel or board, but they may not.

10.1. 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.

10.2. irq – Interrupt Test - …/irq/

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

10.3. regs - Register Access - …/regs/

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

10.4. sbtest - Single Board Test - …/sbtest/

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

> **NOTE**: Multiple instances should not be run simultaneously on the same SIO4.

10.5. syncc2c - SYNC Channel-to-Channel - …/syncc2c/

This application performs an automated test of SYNC transmit and receive functionality on a pair of user specified transmit and receive channels.

10.6. txrate - Transmit Rate - …/txrate/

This application reports the expected transmit clock rate and configuration for a user specific rate.

Document History

Revision	Description
June 13, 2023	Updated to release version 1.59.104.47.0. Minor editorial changes. Updated the description of the open and close calls. Added a note to the Firmware Type Configuration IOCTL service description. Updated information on the Initialize Board and Reset Device service. Updated information on the Channel Reset service. Updated information on the open() call. Updated information on the close() call. Added section and page links to data types definitions where they are used by IOCTL services and other data types.
April 30, 2021	Updated to release version 1.58.93.36.0. Expanded automatic startup information.
March 26, 2021	Updated to release version 1.57.93.36.0. Added notes about multiple instances of the sample applications running simultaneously. Numerous minor editorial changes.
December 9, 2020	Updated to release version 1.57.92.35.0. Updated the kernel support table. Minor editorial changes. Updated the inside cover page. Updated Block Mode DMA macro and associated information. Added a licensing subsection. Expanded automatic startup information.
November 21, 2017	Updated to release version 1.55.73.20.0. Removed library versioning along with the `sync/utils/` code and directory. Directory reorganization. Removed the `synctest` sample application. Removed "`_lib`" from file names. Added section on important files. Numerous editorial modifications.
December 8, 2016	Updated to release version 1.55.69.18.0. Updated the kernel support table. Some document reorganization. Removed the "built" information from the proc file. The build date and time field in the driver information structure is now empty.
March 25, 2014	Updated to release version 1.53.52.0. Added information and support for the Firmware Type Configuration feature. Updated the command line arguments for some of the sample applications. Combined the SYNC and Zilog releases into a single release.
November 15, 2013	Updated to release version 1.52.50.0.
October 10, 2013	Updated to release version 1.51.0. Updated information on the `irq` sample application execution time.
September 25, 2013	Updated to release version 1.50.0. Added cabling notes for the `syncc2c` sample application. Removed the notes about the sample applications not working with the SIO4BXR-SYNC boards. Removed all references to the use of interrupts by the driver itself. Moved driver release version specific notes to this table. Removed the `synctest` sample application.
August 27, 2013	Updated to release version 1.49.0. Updated some of the SIO4_FEATURE_INDEX_* documentation. Updated some of the device index information for the sample applications.
June 29, 2013	Updated to release version 1.48.0. Added several feature query options. Updated the firmware register table. Added a few IOCTL services. Added a few transceiver protocols.
April 17, 2013	Updated to release version 1.47.0. Added documentation for the `SIO4_RX_FIFO_FULL_CFG_CHAN` and `SIO4_RX_FIFO_FULL_CFG_GLB` services.
April 17, 2013	Updated to release version 1.46.0. Renamed the feature option `SIO4_FEATURE_BCR_RX_FFC` to `SIO4_FEATURE_BCR_RX_FFC_GLB`. Added the feature option `SIO4_FEATURE_CSR_RX_FFC_CHAN`.
July 24, 2012	Updated to release version 1.45.0.
May 3, 2012	Updated to release version 1.44.0.
April 13, 2012	Updated to release version 1.43.0. Added numerous options for the Feature Test IOCTL service. Corrected the spelling of the `SIO4_FEATURE_BCR_RX_CFG` Feature Test option. If the FIFO size is unknown, the `SIO4_RX/TX_FIFO_SIZE` and xxx services now return zero. Updated the CPU support data.
January 16, 2012	Updated to release version 1.42.0. Modified the title page model number. Updated the kernel support table. Updated the compiler support information.
August 19, 2011	Updated to release version 1.41.0.
August 11, 2011	Updated to release version 1.40.1.
June 17, 2011	Updated to release version 1.40.0.
March 2, 2011	Updated to release version 1.39.0.

March 1, 2011	Updated to release version 1.38.1. Removed app4. Updated some version notes. Renamed the `app1` application to `synctest`. Renamed the `app2` application to `syncc2c`.
December 11, 2010	Updated to release version 1.38.0. Various editorial changes.
November 22, 2010	Updated to release version 1.37.0. Removed termination configuration from the `sio4_sync_t` structure. Removed all items and services relating to a FIFO's type. Added several Feature Test IOCTL options. Removed the Read FIFO Status IOCTL service. Removed the `app3` sample application and added `sbtest`.
July 27, 2010	Updated to release version 1.36.0. Updated the CPU and Kernel Support information.
June 10, 2010	Updated to release version 1.35.0.
March 18, 2010	Updated to release version 1.33.0.
February 18, 2010	Updated to release version 1.32.1.
February 13, 2010	Updated to release version 1.32.0.
January 25, 2010	Updated to release version 1.31.0. Added the *id* sample application.
December 18, 2009	Updated to release version 1.30.0. Added information regarding the SIO4BXR programmable oscillator feature.
November 12, 2009	Updated to release version 1.29.0.
September 19, 2009	Updated to release version 1.28.0. Updated kernel support list.
September 11, 2009	Updated to release version 1.27.0. Updated kernel support list.
August 23, 2009	Updated to release version 1.26.0. Updated kernel support list. Renamed Overall Make Script. Renamed the driver startup script.
June 2, 2009	Updated to release version 1.25.1.
March 7, 2009	Updated to release version 1.25.0.
February 21, 2009	Updated to release version 1.24.0. Added the sample application `sync/txrate`. Reorganized the installed files sections. Added the `SIO4_FEATURE_FW_PD_BITS` feature test option.
June 25, 2008	Updated to release version 1.23.0. Corrected the names of some IOCTL macros. The accumulated interrupt status is no longer cleared when a new notification request is made. Added information on I/O interrupt usage. Additional kernel porting.
March 29, 2007	Updated to release version 1.22.0. Notes were added for oscillator programming changes applicable to programmable oscillator models.
August 25, 2006	Updated to release version 1.21.0. List specific 2.2, 2.4, 2.6 and 32/64-bit kernels tested.
August 8, 2006	Updated to release version 1.20.0. Added driver updates.
January 30, 2006	Updated to release version 1.19.2. Added an Overall Make Script. Altered the directory structure. As of release this release the directory structure changed to accommodate the asynchronous library code and any associated files and build targets. Only the library sources are included in the SYNC version of the release.
January 25, 2006	Updated to release version 1.19.1.
December 19, 2005	Updated to release version 1.19.0.
October 4, 2005	Updated to release version 1.18.3.
September 30, 2005	Updated to release version 1.18.2.
September 26, 2005	Updated to release version 1.18.1.
July 15, 2005	Updated to release version 1.18.0. Removed feature definitions that are no longer supported.
May 24, 2005	Updated to release version 1.17.1.
May 19, 2005	Updated to release version 1.17.0.
May 10, 2005	Updated to release version 1.16.0. Corrected timeout information. Corrected remarks about the `sio4_sync_tx_t.clock.idle` bit. Added new feature options.
April 5, 2005	Updated to release version 1.15.1.
March 23, 2005	Updated to release version 1.15.0.
January 25, 2005	Updated to release version 1.14.0.
January 24, 2005	Updated to the driver to support the 2.6 kernel.
November 3, 2004	Updated to release version 1.12.1.
November 2, 2004	Updated to release version 1.12.0.
October 18, 2004	Updated to release version 1.11.0.

August 30, 2004	Updated to release version 1.10.0.
August 18, 2004	Updated to release version 1.09.0. Updated documentation on some init and reset services.
August 17, 2004	Updated to release version 1.08.0. Fixed driver `SIO4_INIT_CHANNEL` bug.
August 11, 2004	Updated to release version 1.07.2. Changed UART references to USC.
August 10, 2004	Updated to release version 1.07.1. Added information on supported devices.
August 9, 2004	Updated to release version 1.07.0. Initial driver release.