SIO4/8

Four/Eight Channel High Speed Serial I/O

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

Linux Driver
User Manual

Manual Revision: March 22, 2024
Driver Release Version 3.20.109.50.1

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

Preface

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

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

Table of Contents

Table of Figures

1. Introduction

This release is intended for those SIO4 and SIO8 models which use the Zilog Z16C30 dual USC chips or which are –SYNC models.

> **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.
>
> **NOTE**: The SIO4 contains four independent serial channels. The driver and API Library present each channel as a separate logical device and give separate access to each such device.

1.1. Purpose

The purpose of this document is to describe the interface to the SIO4 API Library and to the underlying Linux device driver. The API Library software provides the interface between "Application Software" and the device driver. The driver software provides the interface between the API Library and the actual SIO4 hardware. The API Library and driver interfaces are based on the board's functionality.

1.2. Acronyms

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

Acronyms	Description
API	Application Programming Interface
BMDMA	Block Mode DMA
DMA	Direct Memory Access
DMDMA	Demand Mode DMA
DPLL	Digital Phase Lock Loop
GSC	General Standards Corporation
HDLC	High-level Data Link Control
PCI	Peripheral Component Interconnect
PCIe	PCI Express
PMC	PCI Mezzanine Card
USC	Universal Serial Controller

1.3. Definitions

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

Term	Definition
…	This is a shortcut representation of the SIO4 installation directory or any of its subdirectories.
API Library	This is a library that provides application-level access to SIO4 hardware.
Application	This is a user mode process, which runs in user space with user mode privileges.
Driver	This is the SIO4 device driver, which runs in kernel space with kernel mode privileges.
Library	This is usually a general reference to the API Library.
Protocol Library	This is a library that provides a programming interface tailored to a specific serial protocol.
SIO4	This is used as a general reference to any board supported by this driver.

1.4. Software Overview

1.4.1. Basic Software Architecture

This section describes the general architecture for the basic components that comprise SIO4 applications. The overall architecture is illustrated in Figure 1 below.

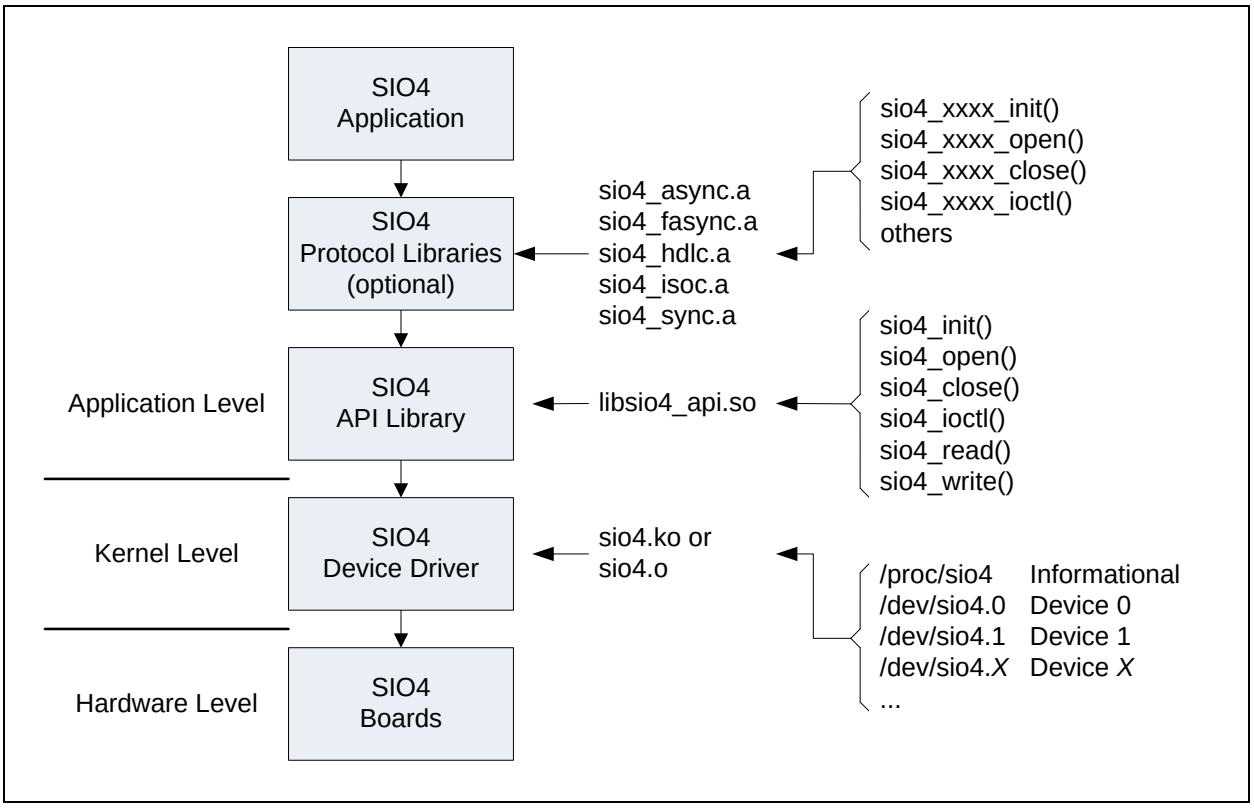

Figure 1 Basic architectural representation.

1.4.2. API Library

The primary means of accessing SIO4 boards is via the SIO4 API Library. This library forms a layer between the application and the driver. Additional information is given in section 3.2.3 (page 18). With the library, applications are able to open and close a device and, while open, perform I/O control and read and write operations.

1.4.3. Protocol Libraries

A secondary means of accessing SIO4 serial channels is via Protocol Libraries. These are serial protocol specific libraries which generally replace the API Library programming interface. Each Protocol Library sits between the application and the API Library. While the Protocol Libraries are intended to replace the API Library programming interface, both can be accessed in parallel when the need arises. Protocol Libraries are provided for the Asynchronous, HDLC, Isochronous and SYNC serial protocols.

1.4.4. Device Driver

The device driver is the host software that provides a means of communicating directly with SIO4 hardware. The driver executes under control of the operating system and runs in Kernel Mode as a Kernel Mode device driver. The driver is implemented as a standard dynamically loadable Linux device driver written in the C programming

language. While applications can access the driver directly without use of the API Library, it is recommended that all access is made through the library.

1.5. Hardware Overview

> **NOTE**: Each SIO8 board presents itself 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. Once the data link between the two devices is established, the desired transfers can be performed and will become transparent to the user. The SIO4 board includes two DMA controllers and comes with a maximum of 256K Bytes of FIFO storage, which is 32K per channel direction (32K * 2 * 4). Each DMA controller is capable of transferring data to and from host memory, whereas the FIFO helps maintain continuous data transfer at the cable interface. 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 comes with transceivers that are fixed as RS232 or RS485/422, or with transceivers that are configurable. The SIO4 comes in two basic varieties; SYNC models or Zilog models, which are based on two Z16C30 dual USC chips. Later model SIO4 boards support both models with the mode being software controlled on a per channel basis. The SIO4 also provides for interrupt generation for various states of the board like Sync Character detection, FIFO empty, FIFO full and DMA complete.

> **NOTE**: Software selection of SYNC or Zilog mode of operation is not at this time explicitly supported by the Protocol Libraries, the SIO4 API Library or the device driver. The operating mode is controlled by the model ordered.

1.6. Reference Material

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

- The *SIO4 Driver Reference Manual* from General Standards Corporation.
- 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.plxteh.com

- The *Z16C30 USC User's Manual* from Zilog. *
- The *Z16C30 Electronic Programmer's Manual* from Zilog (Zilog part number ZEPMDC00001). *

* The Zilog material is available from:

> Zilog, Inc.
> 910 E Hamilton Ave
> CAMPBELL, CA 95008 USA
> Phone: 1-408-558-8500
> WEB: http://www.zilog.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 6.x, 5.x, 4.x, 3.x, 2.6, 2.4 and 2.2 running on a PC system with one or more x86 processors. This release of the driver supports the below listed kernels.

Kernel	Distribution
6.2.9	Red Hat Fedora Core 38
6.0.7	Red Hat Fedora Core 37
5.17.5	Red Hat Fedora Core 36
5.14.10	Red Hat Fedora Core 35
5.11.12	Red Hat Fedora Core 34
5.8.15	Red Hat Fedora Core 33
5.6.6	Red Hat Fedora Core 32
5.3.7	Red Hat Fedora Core 31
5.0.9	Red Hat Fedora Core 30
4.18.16	Red Hat Fedora Core 29
4.16.3	Red Hat Fedora Core 28
4.13.9	Red Hat Fedora Core 27
4.11.8	Red Hat Fedora Core 26
4.8.6	Red Hat Fedora Core 25
4.5.5	Red Hat Fedora Core 24
4.2.3	Red Hat Fedora Core 23
4.0.4	Red Hat Fedora Core 22
3.17.4	Red Hat Fedora Core 21
3.11.10	Red Hat Fedora Core 20
3.9.5	Red Hat Fedora Core 19
3.6.10	Red Hat Fedora Core 18
3.3.4	Red Hat Fedora Core 17
3.1.0	Red Hat Fedora Core 16
2.6.38	Red Hat Fedora Core 15
2.6.35	Red Hat Fedora Core 14
2.6.33	Red Hat Fedora Core 13
2.6.31	Red Hat Fedora Core 12
2.6.29	Red Hat Fedora Core 11
2.6.27	Red Hat Fedora Core 10
2.6.25	Red Hat Fedora Core 9
2.6.23	Red Hat Fedora Core 8
2.6.21	Red Hat Fedora Core 7
2.6.18	Red Hat Fedora Core 6
2.6.15	Red Hat Fedora Core 5
2.6.11	Red Hat Fedora Core 4
2.6.9	Red Hat Fedora Core 3
2.4.18	Red Hat 8.0

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

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

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

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

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

2.1.1. 32-bit Support Under 64-bit Environments

This driver supports 32-bit applications under 64-bit environments. The availability of this feature in the kernel depends on a 64-bit kernel being configured to support 32-bit application compatibility. Additionally, 2.6 kernels prior to 2.6.11 implemented 32-bit compatibility in a way that resulted in some drivers not being able to take advantage of the feature. (In these kernels a driver's IOCTL command codes must be globally unique. Beginning with 2.6.11 this requirement has been lifted.) If the driver is not able to provide 32-bit support under a 64-bit kernel, the "`32-bit support`" field in the `/proc/sio4` file will be "`no`".

2.2. The /proc/ File System

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

While the driver is running, the text file `/proc/sio4` can be read to obtain information about the driver and the boards it detects. Each file entry includes an entry name followed immediately by a colon, a space character, and the entry value. Below is an example of what appears in the file, followed by descriptions of each entry.

```
version: 3.20.109.50
32-bit support: yes
boards: 1
models: SIO4BXR
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 either "`yes`" or "`no`" for 64-bit driver builds and "`yes (native)`" for 32-bit builds.
`boards`	This identifies the total number of boards the driver detected.
`models`	This gives a comma separated list of the basic model number for each board the driver detected. The model numbers are listed in the same order that the boards are accessed via the API Library's open function. For this driver all model numbers should be `SIO4`.
`ids`	This is a comma separated list identifying the values read from the boards' user jumpers.

2.3. File List

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

File	Description
`sio4.linux.tar.gz`	This archive contains the driver, the API Library and all related files.
`sio4_async_rm.pdf`	This is a PDF version of the *Asynchronous Protocol Library Reference Manual*.
`sio4_hdlc_rm.pdf`	This is a PDF version of the *HDLC Protocol Library Reference Manual*.
`sio4_isoc_rm.pdf`	This is a PDF version of the *Isochronous Protocol Library Reference Manual*.
`sio4_linux_um.pdf`	This is a PDF version of this user manual.
`sio4_rm.pdf`	This is a PDF version of the *SIO4 API Library Reference Manual*.
`sio4_sync_rm.pdf`	This is a PDF version of the *SYNC Protocol Library Reference Manual*.

2.4. Directory Structure

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

Directory	Description
sio4/	This is the driver root directory. It contains the documentation, the Overall Make Script (section 2.7, page 14) and the below listed subdirectories.
…/api/	This directory contains the API Library source files (section 3.2.3, page 18).
…/async/	This directory contains the Asynchronous Protoco006C Library and related files.
…/docsrc/	This directory contains the source files for the code samples given in the reference manual (section 6, page 25).
…/driver/	This directory contains the device driver source files (section 5, page 21).
…/hdlc/	This directory contains the HDLC Protocol Library and related files.
…/include/	This directory contains the header files for the various libraries.
…/isoc/	This directory contains the Isochronous Protocol Library and related files.
…/lib/	This directory contains all of the libraries built from the driver archive.
…/samples/	This directory contains the sample application subdirectories and all of their corresponding source files (section 9, page 28).
…/sync/	This directory contains the SYNC Protocol Library and related files.
…/utils/	This directory contains the source files for the utility libraries used by the sample applications (section 7, page 26).

2.5. Installation

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

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

2. Copy the archive file `sio4.linux.tar.gz` into the current directory.

3. Issue the following command to decompress and extract the files from the provided archive. This creates the directory `sio4` in the current directory, and then copies all of the archive's files into this new directory.

   ```
   tar –xzvf sio4.linux.tar.gz
   ```

2.6. Removal

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

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

1. Shutdown the driver as described in section 5.6 (page 24).

2. Change to the directory where the driver archive was installed, which may have been `/usr/src/linux/drivers/`. (The path name may vary among distributions and kernel versions.)

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

   ```
   rm –rf sio4.linux.tar.gz sio4
   ```

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

   ```
   rm –f /dev/sio4.*
   ```

5. If the automatic startup procedure was adopted (section 5.3.2, page 22), then edit the system startup script `rc.local` and remove the line that invokes the SIO4's `start` script. The file `rc.local` should be located in the `/etc/rc.d/` directory.

2.7. Overall Make Script

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

NOTE: The following steps may require elevated privileges.

1. Change to the driver root directory (…`/sio4/`).

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

```
./make_all clean
```

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

```
./make_all
```

2.8. Environment Variables

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

2.8.1. GSC_API_COMP_FLAGS

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

Undefined or Empty	`== Compiling: init.c`
	`== Compiling: ioctl.c`
	`== Compiling: open.c`

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

2.8.2. GSC_API_LINK_FLAGS

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

Undefined or Empty	`==== Linking: ../lib/libsio4_api.so`

Defined and Not Empty	`==== Linking: ../lib/libsio4_api.so (added 'xxx')`

2.8.3. GSC_LIB_COMP_FLAGS

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

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

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

2.8.4. GSC_LIB_LINK_FLAGS

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

Undefined or Empty	`==== Linking: ../lib/sio4_utils.a`
Defined and Not Empty	`==== Linking: ../lib/sio4_utils.a (added 'xxx')`

2.8.5. GSC_APP_COMP_FLAGS

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

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

2.8.6. GSC_APP_LINK_FLAGS

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

Undefined	`==== Linking: id`

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

3. Main Interface Files

This section gives general information on the suggested device interface files to use when developing SIO4 based applications. For additional information refer to the *SIO4 API Library Reference Manual*.

3.1. Main Header File

Throughout the remainder of this document references are made to various header files included as part of the SIO4 driver archive. For ease of use it is suggested that applications include only the single header file shown below rather than individually including those headers identified separately later in this document. Including this header file pulls in all other pertinent SIO4 specific header files. Therefore, sources may include only this one SIO4 header and make files may reference only this one SIO4 include directory.

Description	File	Location
Header File	`sio4_main.h`	`…/include/`

3.2. Main Library File

Throughout the remainder of this document references are made to various statically linkable libraries included with the driver. For ease of use it is suggested that applications link only the single library file shown below rather than individually linking those libraries identified separately later in this document. Linking this library file pulls in all other static libraries included with the driver. Therefore, make files may reference only this one SIO4 static library and only this one SIO4 library directory.

Description	File	Location
Static Libraries	`sio4_main.a`	`…/lib/`
	`sio4_multi.a`	

> **NOTE**: For applications using the SIO4 and no other GSC devices, link the `sio4_main.a` library. For applications using multiple GSC device types, link the `xxxx_main.a` library for one of the devices and the `xxxx_multi.a` library for the others. Linking multiple `xxxx_main.a` libraries may likely produce link errors due to duplicate symbols being defined. While it may make little or no difference, it is recommended that one choose the `xxxx_main.a` library from the driver with the largest number in positions three (x.x.X.x.x) and/or four (x.x.x.X.x) in the driver release version number.
>
> **NOTE**: The SIO4 API Library is implemented as a shared library and is thus not linked with the SIO4 Main Library. The API Library must be linked with applications by adding the linker argument `-lsio4_api` to the linker command line.

3.2.1. Build

The main library is built via the Overall Make Script (section 2.7, page 14). However, the main library can be built separately following the below steps.

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

2. Remove existing build targets using the below command.

   ```
   make clean
   ```

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

   ```
   make
   ```

3.2.2. System Libraries

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

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

3.2.3. Shared Object Script: Build the Main Libraries as Shard Object Files

The main libraries built via the Overall Make Script (section 2.7, page 14) are static library files. Some applications however, require that the Main Libraries be accessed as shared object files. Generating shared object files requires that all of the static libraries be recompiled for this purpose and linked as .so files. This is done using the Shared Object Script named below. When run, the script invokes the Overall Make Script to clean all existing build targets, deletes the two shared object files name below, if they exist, defines an environment variable used by all of the static library make files, invokes the Overall Make Script again to rebuild all existing build targets then invokes make on the library make file (`…/lib/makefile`) to link the shared object files. The required manual steps are as follows.

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

2. Execute the below script.

   ```
   ./static_to_shared.sh
   ```

Running the above named Shared Object Script produces the files given in the table below. These two shared object files fulfill the same purpose as the similarly named static libraries as described in the note under section 3.2 above. Refer to that note when selecting which shared object file to use.

Description	File	Location
Shared Object Files	`libsio4_main.so`	`…/lib/`
	`libsio4_multi.so`	

The shared object files can be linked via two different methods. In the first method, the application linker command line can explicitly name the file in the same manner as is done were it a static library. This is the method used by the sample applications, all of which use the SIO4 API Library, which itself is a shared object file. This file is also found in the `…/lib/` subdirectory. In the second method, the .so files are copied to the `/usr/lib/` subdirectory and are referenced on the application's liker command line as given in the table below.

Library	gcc Link Flag
`libsio4_main.so`	`-lsio4_main`
`libsio4_multi.so`	`-lsio4_multi`

4. API Library

The SIO4 API Library is the software interface between user applications and the SIO4 device driver. The interface is accessed by including the header file `sio4_api.h`.

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

4.1. Files

The library files are summarized in the table below.

Description	File	Location
Source Files	`*.c, *.h, makefile …`	`…/api/`
Header File	`sio4_api.h`	`…/include/`
Library File	`libsio4_api.so`	`…/lib/` `/usr/lib/` *

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

4.2. Build

The API Library is built via the Overall Make Script (section 2.7, page 14), but can be built separately following the below steps.

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

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

2. Remove existing build targets using the below command.

   ```
   make clean
   ```

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

   ```
   make
   ```

4.3. Library Use

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

Description	File	Location	Linker Argument
Header File	`sio4_api.h`	`…/include/`	
Shared Object Library	`libsio4_api.so`	`…/lib/`	
		`/usr/lib/`	`-lsio4_api`

4.4. Macros

For detailed macro information refer to this same section number in the *SIO4 API Library Reference Manual*.

4.5. Data Types

For detailed data type information refer to this same section number in the *SIO4 API Library Reference Manual*.

4.6. Functions

For detailed function information refer to this same section number in the *SIO4 API Library Reference Manual*.

4.7. IOCTL Services

For detailed IOCTL information refer to this same section number in the *SIO4 API Library Reference Manual*.

5. The Driver

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

5.1. Files

The device driver files are summarized in the table below.

Description	Files	Location
Source Files	`*.c`, `*.h`, `Makefile` …	…`/driver/`
Header File	`sio4.h`	
Driver File	`sio4.ko` † `sio4.o` ‡	

† This is for kernel versions 2.6 and later.
‡ This is for kernel versions 2.4 are earlier.

5.2. Build

NOTE: Building the driver requires installation of the kernel headers and possibly other packages.

The device driver is built via the Overall Make Script (section 2.7, page 14), but can be built separately following the below steps.

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

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

   ```
   make clean
   ```

3. Build the driver by issuing the below command.

   ```
   make
   ```

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

5.3. Startup

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

The startup script used in this procedure is designed to load the device driver and create fresh device nodes. This is accomplished by unloading the current driver, if loaded, and then loading the accompanying driver executable. In addition, the script 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 corresponds to the number of boards identified by the driver.

5.3.1. Manual Driver Startup Procedures

Start the driver manually by following the below listed steps.

NOTE: The following steps may require elevated privileges.

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

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

   ```
   ./start
   ```

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

   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 zero, and increase by one for each additional board installed.

3. Verify that the device driver module has been loaded by issuing the below command and examining the output. The module name `sio4` should be included in the output.

   ```
   lsmod
   ```

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

   ```
   ls –l /dev/sio4.*
   ```

5.3.2. Automatic Driver Startup Procedures

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

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

   ```
   /usr/src/linux/drivers/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.

5.3.2.1. File `rc.local` Not Present

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

```
#!/bin/bash

# Add your local content here.
```

5.3.2.2. Default `rc.local` File Permissions

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

5.3.2.3. `systemd` Installations

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

```
systemctl enable rc-local
```

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

5.3.2.4. `systemd` and `rc.local` Timing

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

5.3.2.5. SElinux Implications

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

```
yum install setroubleshoot setools
```

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

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

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

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

5.4. Verification

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

1. Verify that the file `/proc/sio4` is present. If the file is present then the driver is loaded and running. Verify the file's presence by viewing its content with the below command.

   ```
   cat /proc/sio4
   ```

5.5. Version

The driver version number can be obtained in a variety of ways. It is reported by the driver both when the driver is loaded and when it is unloaded (depending on kernel configuration options, this may be visible only in places such as `/var/log/messages`). It is reported in the text file `/proc/sio4` while the driver is loaded and running. The version number is also given in the file `release.txt` in the root install directory.

5.6. Shutdown

Shutdown the driver following the below listed steps.

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

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

   ```
   rmmod sio4
   ```

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

   ```
   lsmod
   ```

6. Document Source Code Examples

This is a library of the source code included in the *SIO4 API Library Reference Manual*. For additional information refer to the *SIO4 API Library Reference Manual*.

6.1. Files

The library files are summarized in the table below.

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

6.2. Build

The library is built via the Overall Make Script (section 2.7, page 14), but can be built separately following the below steps.

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

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

   ```
   make clean
   ```

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

   ```
   make
   ```

4. Rebuild the Main Library (section 3.2.1, page 17).

6.3. Library Use

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

7. Utilities Source Code

The driver archive includes a body of utility source code designed to aid in the understanding and use of the API calls and the IOCTL services. The utility services provide wrappers, mostly visual, around the respective services. Utility sources are also included for device independent and common, general-purpose services. The aim of all the visual wrappers is to facilitate structured console output for the sample applications. The utility services are used extensively by 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. For additional information refer to the *SIO4 API Library Reference Manual*.

7.1. Files

The utility files are summarized in the table below.

Description	Files	Location
Source Files	`*.c`, `*.h`, `makefile` …	…`/utils/`
Header File	`sio4_utils.h`	…`/include/`

Library Files	`sio4_utils.a`
	`gsc_utils.a`
	`os_utils.a`
	`plx_utils.a`
		…`/lib/`

7.2. Build

The libraries are built via the Overall Make Script (section 2.7, page 14), but can be built separately following the below steps.

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

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

   ```
   make clean
   ```

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

   ```
   make
   ```

4. Rebuild the Main Library (section 3.2.1, page 17).

7.3. Library Use

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

8. Operating Information

For operating information refer to this same section number in the *SIO4 API Library Reference Manual*.

9. Sample Applications

The driver archive includes a variety of sample and test applications located under the `samples` subdirectory. While they are provided without support and without any external documentation, any problems reported will be addressed as time permits. The applications are command line based and produce text output for display on a console. All of the applications are built via the Overall Make Script (section 2.7, page 14), but each may be built individually by changing to its respective directory and issuing the commands "`make clean`" and "`make`". 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.

9.1. 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.

9.2. irq - Interrupt Test - …/irq/

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

9.3. led - LED Exerciser - …/led/

This application exercises the board LEDs.

9.4. regs - Register Access - …/regs/

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

9.5. services - Supported IOCTL Services - …/services/

This application reports the set of driver IOCTL services which are supported by the board being accessed. The SIO4 driver includes IOCTL services for virtually all features of every standard model SIO4 produced. This application will identify which of the many services are applicable for the board being accessed.

9.6. txrate - Transmit Bit Rate Calculation - …/txrate/

This application computes the board's necessary settings for user specified bit rates or oscillator frequencies. The configuration and results are reported to the console.

10. Protocol Libraries

An optional means of interacting with SIO4 serial channels is via the Protocol Libraries. These are serial protocol specific libraries which generally replace the API Library programming interface. Each Protocol Library sits between the application and the API Library. While the Protocol Library interfaces generally replace that of the API Library, both can be accessed in parallel when the need arises. For additional information refer to the appropriate Protocol Library reference manual as follows. The files are located in the root install directory.

Protocol	Reference Manual File	Source Code Directory
Asynchronous	sio4_async_rm.pdf	…/sio4/async
HDLC	sio4_hdlc_rm.pdf	…/sio4/hdlc
Isochronous	sio4_isoc_rm.pdf	…/sio4/isoc
SYNC	sio4_sync_rm.pdf	…/sio4/sync

10.1. Files

The location of the Protocol Library source files is per the table below.

Protocol	Location	Header File (…/include/)	Library (…/lib/)
Asynchronous	…/async/api/	sio4_async.h	sio4_async.a
HDLC	…/hdlc/api/	sio4_hdlc.h	sio4_hdlc.a
Isochronous	…/isoc/api/	sio4_isoc.h	sio4_isoc.a
SYNC	…/sync/api/	sio4_sync.h	sio4_sync.a

The Protocol Library files for any given protocol are summarized in the table below.

File	Description
…/<protocol>/*.c	These are protocol specific source files.
…/<protocol>/makefile	This is the library make file.
…/<protocol>/makefile.dep	This is an automatically generated make dependency file.
…/include/sio4_<protocol>.h	This is the Protocol Library header file.
…/lib/sio4_<protocol>.a	This is the Protocol Library static link library file.

10.2. Build

The Protocol Libraries are built via the Overall Make Script (section 2.7, page 14), but can also be built individually as follows.

1. Change to the utility's directory for the desired protocol according to the above table.

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

   ```
   make clean
   ```

3. Compile the source files and build the library using the following command line.

   ```
   make
   ```

4. Rebuild the Main Library (section 3.2.1, page 17).

10.3. Library Use

Each library is used at application compile time and at application link time. At compile time include the appropriate header file given above in each source file using a component of the library interface. Also, edit the include file search path to locate the header file in the above listed directory. At link time include the appropriate static library file listed above with the objects being linked with the application.

11. Protocol Library Utilities

Each of the Protocol Libraries includes a body of utility source code designed to aid in the understanding and use of the API. The utility services provide wrappers, mostly visual, around the respective services. The aim of the visual wrappers is to facilitate structured console output for the sample applications. The utility services are used extensively by 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. For additional information refer to the Protocol Library reference manual as given in the previous section.

11.1. Files

The location of the Protocol Library utility source files is per the table below.

Protocol	**Location**	**Header File (**…/include**)**	**Library (**…/lib/**)**
Asynchronous	…/async/utils/	sio4_async_utils.h	sio4_async_utils.a
HDLC	…/hdlc/utils/	sio4_hdlc_utils.h	sio4_hdlc_utils.a
Isochronous	…/isoc/utils/	sio4_isoc_utils.h	sio4_isoc_utils.a
SYNC	…/sync/utils/	sio4_sync_utils.h	sio4_sync_utils.a

The Protocol Library source files for any given protocol are summarized in the table below.

File	**Description**
…/<protocol>/utils/*.c	These are device specific utility source files.
…/<protocol>/utils/makefile	This is the library make file.
…/<protocol>/utils/makefile.dep	This is an automatically generated make dependency file.
…/include/sio4_<protocol>_utils.h	This is the primary utility header file.
…/lib/sio4_<protocol>_utils.a	This is the statically linkable library file.

11.2. Build

The Protocol Library utility libraries are built via the Overall Make Script (section 2.7, page 14), but can be built separately following the steps below.

1. Change to the directory for the desired protocol according to the tables above.

2. Remove existing build targets using the following command.

   ```
   make clean
   ```

3. Compile the source files and build the library using the following command.

   ```
   make
   ```

4. Rebuild the Main Library (section 3.2.1, page 17).

11.3. Library Use

Each library is used at application compile time and at application link time. At compile time include the above listed header file in each source file using a component of the library interface. Also, edit the include search path to locate the header file in the above listed directory. At link time include the above listed static link library with the objects being linked with the application. Also, edit the library search path to locate the static link library in the above listed directory.

12. Protocol Library Sample Applications

The Protocol Libraries include protocol specific sample applications. While they are provided without support and without any external documentation, any problems reported will be addressed as time permits. The applications are command line based and produce text output for display on a console. All of the applications are built via the Overall Make Script (section 2.7, page 14), 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.

The sample applications are located per the below table.

Protocol	Location
Asynchronous	…/async/samples/
HDLC	…/hdlc/samples /
Isochronous	…/isoc/samples /
SYNC	…/sync/samples /

12.1. Asynchronous

The sample applications for the Asynchronous Protocol Library are located under the …/async/samples/ directory and are built separately as described above.

12.1.1. asyncc2c - Asynchronous Channel-to-Channel Data Transfer - …/asyncc2c/

This application sends data from a specified transmit channel to a specified receive channel using the Asynchronous serial protocol. The specified channels can be on different boards, on the same board or they can even be the same channel. The channels are configured for use of a gender changer, or, when using the same channel, loopback mode.

12.2. HDLC

The sample applications for the HDLC Protocol Library are located under the …/hdlc/samples/ directory and are built separately as described above.

12.2.1. hdlcc2c - HDLC Channel-to-Channel Data Transfer - …/hdlcc2c/

This application sends data from a specified transmit channel to a specified receive channel using the HDLC serial protocol. The specified channels can be on different boards, on the same board or they can even be the same channel. The channels are configured for use of a gender changer, or, when using the same channel, loopback mode.

12.3. Isochronous

The sample applications for the Isochronous Protocol Library are located under the …/isoc/samples/ directory and are built separately as described above.

12.3.1. isocc2c - Isochronous Channel-to-Channel Data Transfer - …/isocc2c/

This application sends data from a specified transmit channel to a specified receive channel using the Isochronous serial protocol. The specified channels can be on different boards, on the same board or they can even be the same channel. The channels are configured for use of a gender changer, or, when using the same channel, loopback mode.

12.4. SYNC

The sample applications for the SYNC Protocol Library are located under the …/sync/samples/ directory and are built separately as described above.

12.4.1. gpio - GPIO Demonstration - …/gpio/

This application demonstrates use of the SYNC cable signals for General Purpose I/O.

12.4.2. syncc2c - SYNC Channel-to-Channel Data Transfer - …/syncc2c/

This application sends data from a specified transmit channel to a specified receive channel using the SYNC serial protocol. The specified channels can be on different boards, on the same board or they can even be the same channel. The channels are configured for use of a gender changer, or, when using the same channel, loopback mode.

Document History

Revision	Description
March 22, 2024	Updated to version 3.20.109.50.1. Added the Shared Object Script section.
March 19, 2024	Updated to version 3.20.109.50.0. Updated the kernel support table. Updated for Fast Async (FASYNC) support.
November 16, 2023	Updated to version 3.19.106.49.0. Minor editorial changes.
June 15, 2023	Updated to version 3.18.104.47.0. Updated the kernel support table. Numerous, minor editorial changes.
December 13, 2022	Updated to version 3.17.101.44.0. Corrected the .so file name in section 4.1. Minor editorial updates.
June 2, 2022	Updated to version 3.16.99.40.0. Updated the kernel support table.
February 8, 2022	Updated to version 3.15.96.38.0. Updated the kernel support table.
August 9, 2021	Updated to version 3.14.94.36.0.
July 8, 2021	Updated to version 3.13.93.36.1. Added section on environment variables.
May 5, 2021	Updated to version 3.13.93.36.0. Expanded automatic startup information.
March 25, 2021	Updated to version 3.12.93.36.1.
February 26, 2021	Updated to version 3.12.93.36.0.
February 18, 2021	Updated to version 3.11.93.35.0. Added notes about multiple instances of the sample applications running simultaneously.
October 12, 2020	Updated to version 3.10.91.34.0. Updated the kernel support table. Expanded automatic startup information. Minor editorial changes.
July 30, 2019	Updated to version 3.9.87.28.0. Updated the kernel support table. Added a licensing subsection. Document reorganization. Various minor editorial updates.
March 24, 2019	Updated to version 3.7.82.26.0.
March 15, 2019	Updated to version 3.7.82.26.0. Some document reorganization and rewriting. Minor editorial changes.
October 18, 2018	Updated to version 3.6.81.25.0. Updated the CPU and kernel support section. Minor editorial changes. Updated the inside cover page. No access mode mods. Added the BMDMA acronym. Minor editorial changes.
October 31, 2017	Updated to version 3.5.73.20.0. Performed some directory reorganization.
December 7, 2016	Updated to version 3.4.69.18.0. Updated the operating information section. Made various miscellaneous updates. Some document reorganization.
October 6, 2016	Updated to version 3.3.67.17.0. Updated the kernel support table.
September 16, 2016	Updated to version 3.2.67.16.0. Updated the command line arguments for the `services` and `txrate` sample applications. Organized the sample applications alphabetically. Updated the kernel support table. Added `-e` and `-i` command line arguments to the channel-to-channel sample application. Added new arguments for the `asyncc2c`, `isocc2c` and `syncc2c` sample applications.
April 13, 2016	Updated to version 3.1.65.13.0. Removed the `built` field from the `/proc/` file. Updated the kernel support table.
March 11, 2016	Updated to version 3.0.59.7.1.
September 7, 2015	Updated to version 3.0.59.7.0.
December 9, 2014	Updated to version 2.8.47.6.1. Added support for the Asynchronous Protocol Library.
December 4, 2014	Updated to version 2.8.47.6.0. The operations section now directs readers to the reference manual. Added support for the Isochronous Protocol Library. All occurrences of “isoch” have been changed to “isoc”, both lower and upper case, including file names. Updated some command line argument descriptions.
July 31, 2014	Updated to version 2.7.44.5.0. Added the `sync/gpio/` sample application.
May 17, 2014	Updated to version 2.6.44.5.0.
April 16, 2014	Updated to version 2.5.43.4.0. Added the Firmware Type Configuration query option. Updated the kernel support table.
October 22, 2013	Updated to version 2.4.42.3.1.

October 22, 2013	Updated to version 2.4.42.3.0.
October 15, 2013	Updated to version 2.3.42.2.0.
August 27, 2013	Updated to version 2.2.42.2.0. Removed the `sync/libtest/` application. Included a preliminary version of the HDLC Protocol Library. Rewrote `txrate` and moved to from the `sync` directory.
October 11, 2012	Updated to version 2.1.41.1.0. Updated the `irq` sample application information.
September 9, 2012	Initial release of the 2.x series driver. A combined release now supports both the Zilog and the –SYNC boards.