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

RTX
Device Driver
User Manual

Manual Revision: November 5, 2015
Driver Release Version 3.1.64.0.0

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

Preface

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

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

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

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

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

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

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

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

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

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

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

Zilog and Z16C30 are trademarks of Zilog, Inc.

Table of Contents

Table of Figures

1. Introduction

1.1. Purpose

The purpose of this document is to provide high level information about the SIO4 RTX device driver. This software provides the application level interface to SIO4 serial channels.

1.2. Acronyms

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

Acronyms	Description
DMA	Direct Memory Access
DMDMA	Demand Mode DMA
GSC	General Standards Corporation
USC	Universal Serial Controller

1.3. Definitions

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

Term	Definition
Application	Application means the user mode process, which runs in the user space with user mode privileges.
Driver	This is the RTX application that communicates directly with the SIO4 hardware.
Root Device	This is the pseudo device accessed by device index `-1`. This is accessed to retrieve basic information about the driver and the devices it has detected.
SIO4	This is used as a general reference to any board supported by this driver.

1.4. Limitations and Restrictions

1.4.1. Changing Time

If the host does not support a high resolution performance counter, then the driver should be restarted any time the `CLOCK_2` clock time is changed.

1.5. Software Overview

The SIO4 RTX driver was developed under the 32-bit version of Windows 7 Ultimate with Service Pack 1. It is written in C using Microsoft Visual Studio 2010 Professional. The driver executable, `sio4.rtss`, is an RTX 2012 application developed with RTX installed into Visual Studio. The driver is accessed by a DLL, which is `sio4_lib.dll` for Windows applications and `sio4_lib.rtdll` for RTX applications. The DLL provides a communications link between the application and the driver executable. The driver provides access to each SIO4 channel independently. The channels are numbered sequentially beginning at zero. The details of the driver interface are documented in the *SIO4 Driver Reference Manual*. The device interface includes open, close, read, write and IOCTL type services. The driver also provides a minimal interface to a pseudo device, referred to here the driver root device, by which an application can determine basic information about the driver and any installed SIO4 boards. The root device interface includes open, close and read type services only. The information returned by root device read requests includes the driver version number, the number of installed SIO4 boards, and the base model number and user jumper identifier values of each board. For more information refer to section 4.6.2 on page 15.

1.6. Hardware Overview

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

The SIO4 is a four channel high-speed serial interface I/O board. This board provides bi-directional serial data transfers between two computers, or a computer and an external peripheral. Each SIO4 board includes two DMA controllers 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. 256K, 32K or 4K). The SIO4 comes with transceivers that are either fixed (RS232 or RS485/422) or are configurable by software. The SIO4 comes in two basic models, one based on a pair of Universal Serial Controllers (Zilog Z16C30 USC) and one based on firmware designed for SYNC style cable interface. The DMA controllers are 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 Sync Character detection, FIFO empty, FIFO full and DMA complete.

1.7. Reference Material

The following reference material may be of particular benefit in using the SIO4 and this driver. 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.plxtech.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: Thttp://www.zilog.com/

2. Installation

2.1. File Extraction

The SIO4 RTX device driver is distributed as a compressed archive. The default file name is `sio4.rtx.zip`, though the distributed file may include the driver version number. The archive includes all source code and associated files for the device driver, the interface library, the Protocol Libraries, the utility libraries, the sample applications and the documentation. The archive should be decompressed and the contents placed in a suitable location. A default location is as given below. This is the default that will be used throughout this manual.

```
c:\gsc\sio4.rtx\
```

2.2. Documentation

The documentation is placed in the root destination directory. The following table briefly describes the documentation provided.

File	Description
`sio4_rtx_um.pdf`	This is a PDF version of this user manual.
`sio4_rm.pdf`	This is a PDF version of the driver Reference Manual.
`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 Isocronous Protocol Library Reference Manual.
`sio4_sync_rm.pdf`	This is a PDF version of the SYNC Protocol Library Reference Manual.

2.3. Directory Structure

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

Directory	Content
`sio4.rtx`	This is the driver's root directory.
`sio4.rtx/batch_files`	This directory contains batch files for building all project target files.
`sio4.rtx/docsrc`	This directory contains the C sources from this user manual.
`sio4.rtx/driver`	This directory contains the driver executable and its sources.
`sio4.rtx/id`	This directory contains the `id` sample application.
`sio4.rtx/include`	This directory contains application level header file. All application level SIO4 headers are located here.
`sio4.rtx/irq`	This directory contains the `irq` sample application.
`sio4.rtx/led`	This directory contains the `led` sample application.
`sio4.rtx/regs`	This directory contains the `regs` sample application.
`sio4.rtx/root`	This directory contains the `root` sample application.
`sio4.rtx/services`	This directory contains the `services` sample application.
`sio4.rtx/txrate`	This directory contains the `txrate` sample application.
`sio4.rtx/utils`	This directory contains utility sources used by the sample applications.
`sio4.rtx/async/asyncc2c`	This directory contains the Asynchronous Channel-to-Channel data transfer sample application.
`sio4.rtx/async/lib`	This directory contains the Asynchronous model SIO4 library sources.
`sio4.rtx/async/utils`	This directory contains Asynchronous specific utility sources.
`sio4.rtx/hdlc/lib`	This directory contains the HDLC model SIO4 library sources. *
`sio4.rtx/hdlc/utils`	This directory contains HDLC specific utility sources. *
`sio4.rtx/isoc/asyncc2c`	This directory contains the Isochronous Channel-to-Channel data transfer sample application.
`sio4.rtx/isoc/lib`	This directory contains the Isochronous model SIO4 library sources.

sio4.rtx/isoc/utils	This directory contains Isochronous specific utility sources. *
sio4.rtx/sync/docsrc	This directory contains the C sources from the SYNC Library user manual.
sio4.rtx/sync/gpio	This directory contains the gpio sample application for the SYNC model of the SIO4.
sio4.rtx/sync/lib	This directory contains the SYNC model SIO4 library sources.
sio4.rtx/sync/syncc2c	This directory contains the syncc2c sample application for the SYNC model of the SIO4.
sio4.rtx/sync/utils	This directory contains SYNC specific utility sources.

* This release includes a preliminary version of the HDLC Protocol Library.

2.4. Overall Build Batch Files

The driver archive includes a set of batch files designed to produce fresh builds of all content included in the archive. The batch files are designed to invoke the build tools included with Visual Studio 2010 as installed in their default location. If your installation of Visual Studio 2010 is not in its default location, then you will have to modify the main Visual Studio 2010 specific batch file to specify the location that is correct for your installation. Refer to the batch file `c:\gsc\sio4.rtx\batch_files\mvs10\_bmake_main.bat`. Before attempting to invoke the batch files, please exit any IDE or editor accessing any of the included project or source files. Follow the below instructions to perform an overall build.

1. In a Command Window, change to the SIO4 batch files directory.

   ```
   cd c:\gsc\sio4.rtx\batch_files
   ```

2. To perform a *clean* of all build targets execute the below command. This should take just a few minutes. Review the output to the screen for problems.

   ```
   .\clean.bat
   ```

3. To build debug versions of all build targets execute the below command. This should take quite a bit of time to finish. Review the output to the screen for problems.

   ```
   .\debug.bat
   ```

4. To build release versions of all build targets execute the below command. This should take quite a bit of time to finish. Review the output to the screen for problems.

   ```
   .\release.bat
   ```

3. Basic Software Architecture

This section describes the general architecture for the basic components that comprise a Windows application using the SIO4 RTX device driver. The overall architecture is illustrated in Figure 1 below.

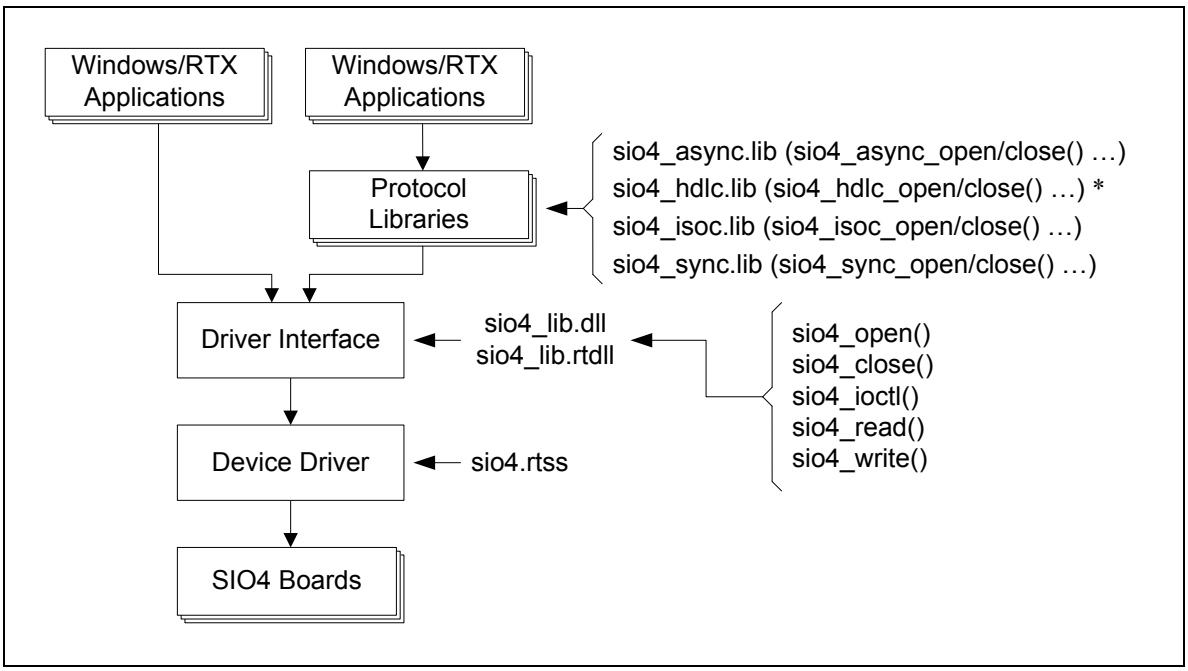

Figure 1 The basic software architecture of applications using the RTX driver.

* This release includes a preliminary version of the HDLC Protocol Library.

3.1. Windows and TRX Applications

At the top of Figure 1 are the applications written to interact with SIO4 boards. These may be customer applications or the sample applications included with the driver archive. Any number of applications may use the driver and libraries simultaneously. However, the driver limits access to each serial channel to one application at a time. Applications are free to access the driver interface library (`sio4_lib.lib`) directly or use one of the provided protocol libraries. While not shown applications may use both interfaces simultaneously.

3.2. Protocol Libraries

Protocol Libraries are libraries whose interface is designed to facilitate the use of specific serial protocols. The interface provided by each library is tailored around the features and functionality specific to the respective protocol. Each library has its own interface, macros and services, which replace those of the device driver itself. The protocol libraries sit on top of the device driver interface and each translates all calls into the library with reciprocal calls into the driver. Each protocol library has its own interface header file and statically linked library. Examples are `sio4_hdlc.h` and `sio4_hdlc.lib`, respectively, for the HDLC serial protocol.

3.3. Device Driver

The device driver is an RTX application, `sio4.rtss`, written specifically to provide a software mechanism for interfacing with installed SIO4 hardware. Access to the driver is through an interface DLL, which implements a communications protocol for exchanging commands and data between applications and the RTX driver executable. In addition to accessing SIO4 hardware, the device driver provides access to the driver root device, a pseudo device,

to provide basic information about the driver and the installed boards. That information includes the driver version number, the number of detected boards, their types, and their jumper id settings.

The driver interface is defined in the header file `sio4_lib.h` and is limited to the services listed in Figure 1. The `sio4_ioctl()` service is the call used to apply changes to or read the setting for a serial channel's numerous parameters. The definitions for all parameters and their options are defined in the driver header files `sio4.h` and `sio4_usc.h`. Refer to the driver reference manual for a complete description of the parameters and their options.

3.4. SIO4 Boards

At the bottom of Figure 1 are the SIO4 boards. Any number of boards may be utilized. While the end product of an effort may be a software application written for a specific model of SIO4, the driver and protocol libraries are written to function with all Zilog and SYNC model boards.

4. Device Driver

The paragraphs that follow give instructions on building, starting, stopping and using the device driver.

4.1. Build

Follow the below steps to build the driver.

1. Load the Visual Studio solutions file listed below.

   ```
   c:\gsc\sio4.rtx\driver\mvs10\sio4.sln
   ```

2. Select the desired application configuration to be built, which must be either “RTSSDebug” or “RTSSRelease”. (The “Debug” and “Release” configurations are not supported.) Do so with the Configuration Manager, which is accessed with the Visual Studio menu selection “Build | Configuration Manager…” In the “Configuration Manager” window that appears, under the “Active solution configuration:” list, select either the “RTSSDebug” or the “RTSSRelease” option. Then click the “Close” button.

3. Remove all existing build targets by selecting the Visual Studio menu option “Build | Clean Solution”. The output in the status area should indicate one success, no failures and nothing skipped.

4. Rebuild all build targets by selecting the Visual Studio menu option “Build | Rebuild Solution”. The output in the status area should indicate no errors, no warnings, one success, no failures and nothing skipped. The end product of the build is the driver executable, as given below. (The debug build targets are located under the respective RTSSDebug directory.)

   ```
   c:\gsc\sio4.rtx\driver\mvs10\RTSSRelease\sio4.rtss
   ```

4.2. Host Preparation

Before the SIO4 RTX driver can be started there are several preparatory steps which must be performed first.

4.2.1. Windows Driver Installation

The SIO4 Windows driver must be installed. This step is necessary so that appropriate system resources are assigned to each board. Each board’s resources include memory regions, an I/O region and an interrupt. The assignment of the resources to each SIO4 is performed by the Windows driver. Installation of the Windows driver is described in documentation specific to that driver. That driver can be downloaded from the General Standards web site.

4.2.2. RTX Device Assignment

Use of the SIO4 RTX driver on the local host requires that all installed SIO4 boards be redirected for use by RTX rather than Windows. For instructions on this procedure refer to the IntervalZero web site. Once at their wed site search for a section titled “Converting a Windows Device to an RTX Device”. Perform the steps described before continuing.

4.3. Startup

Follow the below steps to start driver execution.

1. From a Command Window enter the below command to start the driver.

   ```
   cd c:\gsc\sio4.rtx\driver\mvs10\RTSSRelease\
   rtssrun sio4.rtss
   ```

2. The driver should start and generate a small amount of display output, which is sent to the RtxServer console. The driver will continue to execute until told to exit.

   NOTE: During driver startup the display output will include the basic model number of each discovered board. The output will appear as "`sio4: device loaded: SIO4BX`", presuming, for example, the board is a model SIO4BX.

   NOTE: At the end of driver startup the last line of the displayed output will indicate the status of the effort. The last line will start with "`sio4: driver loaded:`" if the effort was successful. In this case the driver will remain running and provide access to the boards discovered. The last line will start with "`sio4: driver load failure:`" if the effort was not successful. In this case the driver immediately exits, preventing access to installed boards.

4.4. Shutdown

Shutdown the driver using the below options.

4.4.1. Shutdown using driver command line arguments.

Follow the below steps to tell the driver to exit.

1. From a Command Window enter the below command to terminate the driver.

   ```
   rtssrun c:\gsc\sio4.rtx\driver\RTSSRelease\sio4.rtss -x
   ```

Argument	Description
`-h`	Display usage/help information.
`-x`	This initiates driver exit.

2. The driver instance being executed should take only a second or so to complete its task. The driver already running should exit shortly thereafter.

4.5. Version

The driver version number can be obtained in two ways. First, it is reported by the driver both when the driver is loaded and when it is unloaded. This is part of the display output generated by the driver. Second, it is reported in the text obtained by reading from the driver root device, which is accessed using the driver interface call `sio4_open(-1)`.

4.6. Access

4.6.1. Device Access: index zero and above

The SIO4 driver provides individual access to each SIO4 serial channel. Using the driver interface (section 5, page 16) an application calls `sio4_open(`*x*`)` to access a serial channel, where the "x" in the call corresponds directly to the zero based device index.

> **NOTE:** If a serial channel is in an open state when an application exits, the serial channel will remain in the opened state. If this occurs the driver may have to be restarted to gain subsequent access to the same channel.

4.6.2. Driver Root Device: index -1

The driver provides basic driver and device information by accessing the driver root device, which is a pseudo device. This pseudo device is maintained so that applications can gain information about the driver and installed devices without having to query individual devices. (Such queries may not be possible as the driver restricts device access to only one application at a time.) The root device is accessed using the `sio4_open(-1)` call from the device driver. Once opened, a read returns the below information.

```
version: 3.1.64.0
boards: 1
models: SIO4BX
ids: 0x3
```

Entry	Description
`version`	This gives the driver version number in the form `x.x.x.x`.
`boards`	This identifies the total number of SIO4 boards the driver detected.
`models`	This is a list that identifies the base model numbers of the boards detected by the driver. The list is in the order the boards were detected by the driver. If the driver cannot specifically identify a board's type it will be listed only as "SIO4".
`ids`	This is a list identifying the values read from the boards' user jumpers. The list is in the order the boards were detected by the driver.

4.7. Driver Interface

The functional interface to the driver is IOCTL based. The IOCTL services are utilized in combination to achieve a higher level of functionality. The IOCTL command codes are defined in the header files `sio4.h` and `sio4_usc.h`. Their default location is `c:\gsc\sio4.rtx\include`. The IOCTL command codes are documented in the *SIO4 Driver Reference Manual* (`sio4_rm.pdf`).

An alternative to relying strictly on the IOCTL services is to use one of the Protocol Libraries designed to support a specific serial protocol. These Protocol Libraries provide an interface tailored for use of the corresponding protocol. The libraries provide a higher level interface that encapsulates all the IOCTL service calls necessary for proper configuration and use of that protocol with the SIO4. The Protocol Libraries don't prevent the use of the IOCTL services, nor do they preclude their use. What the libraries do is significantly reduce the need for their use as well as significantly reduce the learning curve when using an SIO4. For additional information refer to the specific Protocol Library documentation.

4.8. Driver Use

The driver library is usable by referencing a component of its interface in application sources, including the header file in those sources, and linking the driver interface library with the application. Inclusion of the header (`sio4_lib.h`) is usually done by naming the header file without the path, and then adding the path (`c:\gsc\sio4.rtx\include`) to the compiler's search path list. Inclusion of the library can be done in like fashion with the linker, or by including the entire path and file name together without adjusting the linker search path. For debug builds the driver library is `c:\gsc\sio4.rtx\lib\mvs10\Debug\sio4_lib.lib`. For release builds the driver library is `c:\gsc\sio4.rtx\lib\mvs10\Release\sio4_lib.lib`. At execution, Windows must be able to locate the driver interface DLL, which is `sio4_lib.dll`. This file can be copied from its original location to the same directory as the application, or, using the preferred method of copying the file to the Window's system32 directory (usually `c:\windows\system32`).

5. Driver Interface

The driver interface is accessed by the below header file and includes the services described in the following paragraphs.

```
c:\gsc\sio4.rtx\include\sio4_lib.h
```

> **NOTE:** If an application leaves a channel open when it terminates, then the channel may be left in an open state. This will prevent subsequent access to the channel until the driver is reloaded.

5.1.1. sio4_open()

This function opens communication with the driver to a specified serial channel. Once opened the channel must be closed before it can be accessed by another application. The driver does not permit simultaneous access to a single channel by multiple applications.

> **NOTE:** If a serial channel is in an open state when an application exits, the serial channel will remain in the opened state. If this occurs the driver may have to be reloaded to gain subsequent access to the same channel.

> **NOTE:** There are no restrictions on the number of simultaneous accesses to the root device, which is accessed via `sio4_open(-1)`.

Prototype

```
int sio4_open(int index);
```

Argument	Description
`index`	This is the zero based index of the board to access. Values zero and above will access serial channels. The value `-1` will access the root device.

Return Value	Description
`< 0`	An error occurred. Refer to `errno.h`.
`> 0`	The operation succeeded. Use this value for all future accesses until the device is closed.

Example

```
#include <stdio.h>

#include "sio4_dsl.h"

int sio4_dsl_open(int index)
{
    int fd;

    fd  = sio4_open(index);

    if (fd < 0)
        printf("ERROR: sio4_open(%d) failure\n", index);

    return(fd);
}
```

5.1.2. sio4_close()

This function ends previously established communication with the driver to a serial channel. Once closed the serial channel becomes available for use by other applications.

> **NOTE:** A request to close access to a channel may return a failure status thought access to the device may have been successfully closed.

Prototype

```
int sio4_close(int fd);
```

Argument	Description
fd	This is the file descriptor used to access the device. This is the value previously returned by `sio4_open()`. The file descriptor is not valid after the close operation.

Return Value	Description
< 0	An error occurred. Refer to `errno.h`.
0	The operation succeeded.
> 0	An error occurred. Refer to Windows error codes.

Example

```
#include <stdio.h>

#include "sio4_dsl.h"

int sio4_dsl_close(int fd)
{
    int errs;
    int ret;

    ret = sio4_close(fd);

    if (ret < 0)
        printf("ERROR: sio4_close() failure\n");

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

5.1.3. sio4_ioctl()

This function is the entry point to performing I/O control operations (IOCTL operations). The operations recognized by the driver are defined in header files `sio4.h` and `sio4_usc.h` (both header files can be found in the directory `c:\gsc\sio4.rtx\include` and are automatically included with `sio4_lib.h`). The operations supported by any given SIO4 board can be determined by using the `services` sample application (see section 8.6, page 32). All IOCTL requests on a given serial channel are serialized and will be processed in the order received by the driver. This is a blocking call, though most requests will return almost immediately. The only exception is the Wait Event service (`SIO4_IOCTL_WAIT_EVENT`). IOCTL service requests are not blocked while threads sleep as part of the Wait Event service.

Prototype

```
int sio4_ioctl(int fd, int cmd, void* arg);
```

Argument	Description
fd	This is the file descriptor used to access the device. This is the value previously returned by sio4_open().
cmd	This is one of the IOCTL command codes defined in the header files sio4.h and sio4_usc.h.
arg	This is a pointer to the variable or structure being passed to the driver. The data type is specific to the IOCTL command code being used.

Return Value	Description
< 0	An error occurred. Refer to errno.h.
0	The operation succeeded.
> 0	An error occurred. Refer to Windows error code.

Example

```
#include <stdio.h>

#include "sio4_dsl.h"

int sio4_dsl_ioctl(int fd, int cmd, void* arg)
{
    int errs;
    int ret;

    ret = sio4_ioctl(fd, cmd, arg);

    if (ret < 0)
        printf("ERROR: sio4_ioctl() failure\n");

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

5.1.4. sio4_read()

This function is the entry point to retrieving data received by an SIO4 serial channel. The call will return either when the data request has been satisfied or when the read I/O timeout expires. Read requests are blocking calls, except when a PIO mode request is made with the read I/O timeout set at zero. See the driver reference manual for additional information. Read requests to a given serial channel are serialized as only one read can be active at a time.

Prototype

```
int sio4_read(int fd, void* dst, int bytes);
```

Argument	Description
fd	This is the file descriptor used to access the device. This is the value previously returned by sio4_open().
dst	This is the destination buffer for the data retrieved from the board.

bytes	This is the maximum number of bytes being requested, and must not exceed the size of the specified destination buffer.

Return Value	Description
< 0	An error occurred. Refer to errno.h.
>= 0	The operation succeeded. The value indicates the number of bytes placed in the destination buffer. A value less than the bytes argument typically means the read I/O timeout expired.

Example

```
#include <stdio.h>

#include "sio4_dsl.h"

int sio4_dsl_read(int fd, void* dst, int bytes)
{
    int ret;

    ret = sio4_read(fd, dst, bytes);

    if (ret < 0)
        printf( "ERROR: sio4_read() failure\n");

    return(ret);
}
```

5.1.5. sio4_write()

This function is the entry point to providing data to be transmitted by an SIO4 serial channel. The call will return either when the data transfer has been completed or when the write I/O timeout expires. Write requests are blocking calls, except when a PIO mode request is made with the write I/O timeout set at zero. See the driver reference manual for additional information. Write requests to a given serial channel are serialized as only one write can be active at a time.

Prototype

```
int sio4_write(int fd, const void* src, int bytes);
```

Argument	Description
fd	This is the file descriptor used to access the device. This is the value previously returned by sio4_open().
src	This is the source buffer for the data being provided to the board.
bytes	This is the maximum number of bytes to send, and must not exceed the size of the specified source buffer.

Return Value	Description
< 0	An error occurred. Refer to errno.h.
>= 0	The operation succeeded. The value indicates the number of bytes taken from the source buffer. A value less than the bytes argument typically means the write I/O timeout expired.

Example

```
#include <stdio.h>

#include "sio4_dsl.h"

int sio4_dsl_write(int fd, const void* src, int bytes)
{
    int ret;

    ret = sio4_write(fd, src, bytes);

    if (ret < 0)
        printf("ERROR: sio4_write() failure\n");

    return(ret);
}
```

6. Protocol Libraries

The Serial Protocol Libraries are statically linked utility libraries that provide serial protocol specific interfaces to the SIO4. The libraries are designed to facilitate use of the respective protocol when communicating with an SIO4. The libraries and their support code are located under the SIO4 root directory as named in the below table. Some of the protocols may also include sample applications and additional documentation sources or utility sources.

Protocol	Item	Name
Asynchronous	Root Directory	`c:\gsc\sio4.rtx\async\`
	Header File	`…\include\sio4_async.h`
	Solution File	`…\lib\mvs10\sio4_async.sln`
	Debug Library	`…\lib\mvs10\Debug\sio4_async.lib`
	Release Library	`…\lib\mvs10\Release\sio4_async.lib`
HDLC	Root Directory	`c:\gsc\sio4.rtx\hdlc\`
	Header File	`…\include\sio4_hdlc.h`
	Solution File	`…\lib\mvs10\sio4_hdlc.sln`
	Debug Library	`…\lib\mvs10\Debug\sio4_hdlc.lib`
	Release Library	`…\lib\mvs10\Release\sio4_hdlc.lib`
Isochronous	Root Directory	`c:\gsc\sio4.rtx\isoc\`
	Header File	`…\include\sio4_isoc.h`
	Solution File	`…\lib\mvs10\sio4_isoc.sln`
	Debug Library	`…\lib\mvs10\Debug\sio4_isoc.lib`
	Release Library	`…\lib\mvs10\Release\sio4_isoc.lib`
SYNC	Root Directory	`c:\gsc\sio4.rtx\sync\`
	Header File	`…\include\sio4_sync.h`
	Solution File	`…\lib\mvs10\sio4_sync.sln`
	Debug Library	`…\lib\mvs10\Debug\sio4_sync.lib`
	Release Library	`…\lib\mvs10\Release\sio4_sync.lib`

6.1. Build

The protocol libraries are each built in a similar manner. For each respective library follow the steps below.

1. Load the Visual Studio solutions file listed in the preceding table.

2. Select the desired application configuration to be built. Do so with the Configuration Manager, which is accessed with the Visual Studio menu selection “Build | Configuration Manager…” In the “Configuration Manager” window that appears, under the “Active solution configuration:” list, select either the “Debug” or the “Release” option. Then click the “Close” button.

3. Remove all existing build targets by selecting the Visual Studio menu option “Build | Clean Solution”. The output in the status area should indicate one success, no failures and nothing skipped.

4. Rebuild all build targets by selecting the Visual Studio menu option “Build | Rebuild Solution”. The output in the status area should indicate no errors, no warnings, one success, no failures and nothing skipped. The final built target is a statically linked library, as named in the preceding table.

> **NOTE:** The HDLC Protocol Library is preliminary and untested. The HDLC open and close services are functional, and required. The services for accessing the HDLC settings should be entirely functional. The HDLC I/O services are known to not be functional at this time. Applications should use the generic SIO4 I/O services instead.

6.2. Library Use

The Protocol Libraries are usable by referencing components of their interface in application sources, including the named header file in those sources, and linking the respective library with the application. Inclusion of the header is usually done by naming the header file without the path, and then adding the path to the compiler's search path list. Inclusion of the library can be done in like fashion with the linker, or by including the entire path and file name together without adjusting the linker search path.

7. Utility Libraries

The driver is accompanied by number of utility libraries whose purpose is to aid in reducing the learning curve when using the SIO4. These include both documentation source libraries that bundle the sample code referenced in the various SIO4 software user manuals, and other utility source designed to facilitate the use of driver and the Protocol Libraries.

7.1. Document Source Code Libraries

The source code given in the various user manuals is generally given in modular form and is usable as is. The source code is grouped according to the document in which it is referenced. The purpose of these libraries is to insure that the examples compile and build. The files for the various document source libraries are listed the table below.

Application	Item	Name
General	Root Directory	`c:\gsc\sio4.rtx\docsrc`
	Header File	`…\include\sio4_dsl.h`
	Solution File	`…\mvs10\sio4_dsl.sln`
	Debug Library	`…\mvs10\Debug\sio4_dsl.lib`
	Release Library	`…\mvs10\Release\sio4_dsl.lib`
SYNC	Root Directory	`c:\gsc\sio4.rtx\sync\docsrc\`
	Header File	`…\include\sio4_sync_dsl.h`
	Solution File	`…\mvs10\sio4_sync_dsl.sln`
	Debug Library	`…\mvs10\Debug\sio4_sync_dsl.lib`
	Release Library	`…\mvs10\Release\sio4_sync_dsl.lib`

7.1.1. Build

The document source libraries are each built in a similar manner. For each respective library follow the steps below.

1. Load the Visual Studio solutions file listed in the preceding table.

2. Select the desired application configuration to be built. Do so with the Configuration Manager, which is accessed with the Visual Studio menu selection “Build | Configuration Manager…” In the “Configuration Manager” window that appears, under the “Active solution configuration:” list, select either the “Debug” or the “Release” option. Then click the “Close” button.

3. Remove all existing build targets by selecting the Visual Studio menu option “Build | Clean Solution”. The output in the status area should indicate one success, no failures and nothing skipped.

4. Rebuild all build targets by selecting the Visual Studio menu option “Build | Rebuild Solution”. The output in the status area should indicate no errors, no warnings, one success, no failures and nothing skipped. The final built target is a statically linked library, as named in the preceding table.

7.1.2. Library Use

The document source libraries are usable by referencing components of their interface in application sources, including the named header file in those sources, and linking the respective library with the application. Inclusion of the header is usually done by naming the header file without the path, and then adding the path to the compiler’s search path list. Inclusion of the library can be done in like fashion with the linker, or by including the entire path and file name together without adjusting the linker search path.

7.2. Utility Source Code Libraries

The driver archive includes utility source code suitable mostly for command line applications. A utility source file is included for every driver IOCTL service and for various other interface services. Their purpose is to permit simple use of each service with suitable display output, if desired, where applicable, along with status. When there is an error, an error message is reported to the screen. Each service may be used to apply a setting, request the current setting, or to request support status. Other utility services are included as well. The files for the various utility source libraries are listed the table below.

Application	Item	Name
General	Root Directory	c:\gsc\sio4.rtx\utils
	Header File	…\include\sio4_utils.h
	Solution File	…\mvs10\sio4_utils.sln
	Debug Library	…\mvs10\Debug\sio4_utils.lib …\mvs10\Debug\gsc_utils.lib
	Release Library	…\mvs10\Release\sio4_utils.lib …\mvs10\Release\gsc_utils.lib
Asynchronous	Root Directory	c:\gsc\sio4.rtx\async\utils\
	Header File	…\include\sio4_async_utils.h
	Solution File	…\mvs10\sio4_async_utils.sln
	Debug Library	…\mvs10\Debug\sio4_async_utils.lib
	Release Library	…\mvs10\Release\sio4_async_utils.lib
HDLC	Root Directory	c:\gsc\sio4.rtx\hdlc\utils\
	Header File	…\include\sio4_hdlc_utils.h
	Solution File	…\mvs10\sio4_hdlc_utils.sln
	Debug Library	…\mvs10\Debug\sio4_hdlc_utils.lib
	Release Library	…\mvs10\Release\sio4_hdlc_utils.lib
Isochronous	Root Directory	c:\gsc\sio4.rtx\isoc\utils\
	Header File	…\include\sio4_isoc_utils.h
	Solution File	…\mvs10\sio4_isoc_utils.sln
	Debug Library	…\mvs10\Debug\sio4_isoc_utils.lib
	Release Library	…\mvs10\Release\sio4_isoc_utils.lib
SYNC	Root Directory	c:\gsc\sio4.rtx\sync\utils\
	Header File	…\include\sio4_sync_utils.h
	Solution File	…\mvs10\sio4_sync_utils.sln
	Debug Library	…\mvs10\Debug\sio4_sync_utils.lib
	Release Library	…\mvs10\Release\sio4_sync_utils.lib

7.2.1. Build

The utility source libraries are each built in a similar manner. For each respective library follow the steps below.

1. Load the Visual Studio solutions file listed in the preceding table.

2. Select the desired application configuration to be built. Do so with the Configuration Manager, which is accessed with the Visual Studio menu selection “Build | Configuration Manager…” In the “Configuration Manager” window that appears, under the “Active solution configuration:” list, select either the “Debug” or the “Release” option. Then click the “Close” button.

3. Remove all existing build targets by selecting the Visual Studio menu option “Build | Clean Solution”. The output in the status area should indicate one success, no failures and nothing skipped.

4. Rebuild all build targets by selecting the Visual Studio menu option “Build | Rebuild Solution”. The output in the status area should indicate no errors, no warnings, one success, no failures and nothing skipped. The final built target is a statically linked library, as named in the preceding table.

7.2.2. Library Use

The utility source libraries are usable by referencing components of their interface in application sources, including the named header file in those sources, and linking the respective library with the application. Inclusion of the header is usually done by naming the header file without the path, and then adding the path to the compiler’s search path list. Inclusion of the library can be done in like fashion with the linker, or by including the entire path and file name together without adjusting the linker search path.

8. Sample Applications

The sample applications were developed under Windows 7 Ultimate, 32-bit. All of the applications are written in C using Microsoft Visual Studio 2010 Professional. The sample applications are all Windows executables (.exe files) written to access an SIO4 using the RTX based SIO4 device driver. All project files include Debug and Release configurations for 32-bit versions of Windows. Building the applications first requires that the libraries from the previous section be built. Using the applications requires that the driver be built and that it is running.

8.1. root – Driver Root Device

This sample application opens the driver's root device, which is a pseudo device, then dumps its contents to the console. The content includes various pieces of information about the driver and the SIO4 boards it has detected. For additional information refer to section 4.6.2, page 15.

8.1.1. Build

Follow the below steps to build the sample application.

1. Load the Visual Studio solutions file listed below.

   ```
   c:\gsc\sio4.rtx\root\mvs10\root.sln
   ```

2. Select the desired application configuration to be built. Do so with the Configuration Manager, which is accessed with the Visual Studio menu selection "Build | Configuration Manager…" In the "Configuration Manager" window that appears, under the "Active solution configuration:" list, select either the "Debug" or the "Release" option. Then click the "Close" button.

3. Remove all existing build targets by selecting the Visual Studio menu option "Build | Clean Solution". The output in the status area should indicate one success, no failures and nothing skipped.

4. Rebuild all build targets by selecting the Visual Studio menu option "Build | Rebuild Solution". The output in the status area should indicate no errors, no warnings, one success, no failures and nothing skipped.

8.1.2. Execute

Follow the below steps to execute the sample application, which should have been built using the above procedure.

1. Using Visual Studio, the easiest way to run the program is to select the menu option "Debug | Start Debugging". Any command line options can be set in the project's Property Pages.

2. The application can also be started from a command prompt as given below. For a release build, enter "`Release`" instead "`Debug`". A single iteration may take a few second to complete. Enter command line arguments per the table below.

   ```
   cd c:\gsc\sio4.rtx\root\mvs10\Debug\
   .\root.exe
   ```

Argument	Description
	This application has no command line arguments.

8.2. id - Identify Board

This sample console application provides a command line driven application that provides detailed board identification information. This can be used with tech support to help identify as much technical information about the board as possible from software.

8.2.1. Build

Follow the below steps to build the sample application.

1. Load the Visual Studio solutions file listed below.

   ```
   c:\gsc\sio4.rtx\id\mvs10\id.sln
   ```

2. Select the desired application configuration to be built. Do so with the Configuration Manager, which is accessed with the Visual Studio menu selection "Build | Configuration Manager…" In the "Configuration Manager" window that appears, under the "Active solution configuration:" list, select either the "Debug" or the "Release" option. Then click the "Close" button.

3. Remove all existing build targets by selecting the Visual Studio menu option "Build | Clean Solution". The output in the status area should indicate one success, no failures and nothing skipped.

4. Rebuild all build targets by selecting the Visual Studio menu option "Build | Rebuild Solution". The output in the status area should indicate no errors, no warnings, one success, no failures and nothing skipped.

8.2.2. Execute

Follow the below steps to execute the sample application, which should have been built using the above procedure.

1. Using Visual Studio, the easiest way to run the program is to select the menu option "Debug | Start Debugging". Any command line options can be set in the project's Property Pages.

2. The application can also be started from a command prompt as given below. For a release build, enter "`Release`" instead "`Debug`". A single iteration may take a few second to complete. Enter command line arguments per the table below.

   ```
   cd c:\gsc\sio4.rtx\id\mvs10\Debug\
   .\id.exe
   ```

Argument	Description
`index`	This is the zero based index of the board to access.

8.3. irq – Interrupt Test

This sample application provides a command line driven application intended to verify the operation of the board's interrupts.

8.3.1. Build

Follow the below steps to build the sample application.

1. Load the Visual Studio solutions file listed below.

   ```
   c:\gsc\sio4.rtx\irq\mvs10\irq.sln
   ```

2. Select the desired application configuration to be built. Do so with the Configuration Manager, which is accessed with the Visual Studio menu selection "Build | Configuration Manager…" In the "Configuration Manager" window that appears, under the "Active solution configuration:" list, select either the "Debug" or the "Release" option. Then click the "Close" button.

3. Remove all existing build targets by selecting the Visual Studio menu option "Build | Clean Solution". The output in the status area should indicate one success, no failures and nothing skipped.

4. Rebuild all build targets by selecting the Visual Studio menu option "Build | Rebuild Solution". The output in the status area should indicate no errors, no warnings, one success, no failures and nothing skipped.

8.3.2. Execute

Follow the below steps to execute the sample application, which should have been built using the above procedure.

1. Using Visual Studio, the easiest way to run the program is to select the menu option "Debug | Start Debugging". Any command line options can be set in the project's Property Pages.

2. The application can also be started from a command prompt as given below. For a release build, enter "`Release`" instead "`Debug`". Each iteration should take well under one minute. Enter command line arguments per the table below.

   ```
   cd c:\gsc\sio4.rtx\irq\mvs10\Debug\
   .\irq.exe
   ```

Argument	Description
`-c`	Perform continuous testing until an error occurs.
`-C`	Perform continuous testing, continuing even if errors occur.
`-m#`	Limit continuous testing to this many minutes.
`-n#`	Limit continuous testing to this number of iterations.
`index`	This is the zero based index of the channel to access.

8.4. led – LED Test

This sample application provides a command line driven application that exercises the software accessible LEDs on the SIO4. The purpose of this application is to provide a working example of how to control the LEDs.

8.4.1. Build

Follow the below steps to build the sample application.

1. Load the Visual Studio solutions file listed below.

   ```
   c:\gsc\sio4.rtx\root\mvs10\root.sln
   ```

2. Select the desired application configuration to be built. Do so with the Configuration Manager, which is accessed with the Visual Studio menu selection "Build | Configuration Manager…" In the "Configuration Manager" window that appears, under the "Active solution configuration:" list, select either the "Debug" or the "Release" option. Then click the "Close" button.

3. Remove all existing build targets by selecting the Visual Studio menu option "Build | Clean Solution". The output in the status area should indicate one success, no failures and nothing skipped.

4. Rebuild all build targets by selecting the Visual Studio menu option "Build | Rebuild Solution". The output in the status area should indicate no errors, no warnings, one success, no failures and nothing skipped.

8.4.2. Execute

Follow the below steps to execute the sample application, which should have been built using the above procedure.

1. Using Visual Studio, the easiest way to run the program is to select the menu option "Debug | Start Debugging". Any command line options can be set in the project's Property Pages.

2. The application can also be started from a command prompt as given below. For a release build, enter "`Release`" instead "`Debug`". A single iteration should take well under one minute. Enter command line arguments per the table below.

   ```
   cd c:\gsc\sio4.rtx\led\mvs10\Debug\
   .\led.exe
   ```

Argument	Description
`-c`	Perform continuous testing until an error occurs.
`-C`	Perform continuous testing, continuing even if errors occur.
`-m#`	Limit continuous testing to this many minutes.
`-n#`	Limit continuous testing to this number of iterations.
`index`	This is the zero based index of the channel to access.

8.5. regs - Register Access

This sample application is a menu based command line application that permits interactive access to the board's registers, including write access to the GSC and USC specific registers.

8.5.1. Build

Follow the below steps to build the sample application.

1. Load the Visual Studio solutions file listed below.

   ```
   c:\gsc\sio4.rtx\regs\mvs10\regs.sln
   ```

2. Select the desired application configuration to be built. Do so with the Configuration Manager, which is accessed with the Visual Studio menu selection "Build | Configuration Manager…" In the "Configuration Manager" window that appears, under the "Active solution configuration:" list, select either the "Debug" or the "Release" option. Then click the "Close" button.

3. Remove all existing build targets by selecting the Visual Studio menu option "Build | Clean Solution". The output in the status area should indicate one success, no failures and nothing skipped.

4. Rebuild all build targets by selecting the Visual Studio menu option "Build | Rebuild Solution". The output in the status area should indicate no errors, no warnings, one success, no failures and nothing skipped.

8.5.2. Execute

Follow the below steps to execute the sample application, which should have been built using the above procedure.

1. Using Visual Studio, the easiest way to run the program is to select the menu option "Debug | Start Debugging". Any command line options can be set in the project's Property Pages.

2. The application can also be started from a command prompt as given below. For a release build, enter "`Release`" instead "`Debug`". Enter command line arguments per the table below. Select the desired options according to the menus presented. Select the main menu exit option when finished.

   ```
   cd c:\gsc\sio4.rtx\regs\mvs10\Debug\
   .\regs.exe
   ```

Argument	Description
`index`	This is the zero based index of the channel to access.

8.6. services – Supported IOCTL Services

This sample application is a command line application that reports the set of driver IOCTL services which are supported for 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 to the board being accessed.

8.6.1. Build

Follow the below steps to build the sample application.

1. Load the Visual Studio solutions file listed below.

   ```
   c:\gsc\sio4.rtx\services\mvs10\services.sln
   ```

2. Select the desired application configuration to be built. Do so with the Configuration Manager, which is accessed with the Visual Studio menu selection "Build | Configuration Manager…" In the "Configuration Manager" window that appears, under the "Active solution configuration:" list, select either the "Debug" or the "Release" option. Then click the "Close" button.

3. Remove all existing build targets by selecting the Visual Studio menu option "Build | Clean Solution". The output in the status area should indicate one success, no failures and nothing skipped.

4. Rebuild all build targets by selecting the Visual Studio menu option "Build | Rebuild Solution". The output in the status area should indicate no errors, no warnings, one success, no failures and nothing skipped.

8.6.2. Execute

Follow the below steps to execute the sample application, which should have been built using the above procedure.

1. Using Visual Studio, the easiest way to run the program is to select the menu option "Debug | Start Debugging". Any command line options can be set in the project's Property Pages.

2. The application can also be started from a command prompt as given below. For a release build, enter "`Release`" instead "`Debug`". A single iteration should take just a few seconds. Enter command line arguments per the table below.

   ```
   cd c:\gsc\sio4.rtx\services\mvs10\Debug\
   .\services.exe
   ```

Argument	Description
`-u`	List only unsupported services.
`index`	This is the zero based index of the channel to access.

8.7. txrate – Transmit Bit Rate Calculation

This sample console application will compute the board's necessary settings for user specified bit rates or oscillator frequencies. The calculations are made according to the command line arguments. The configuration and the results are reported as output.

8.7.1. Build

Follow the below steps to build the sample application.

1. Load the Visual Studio solutions file listed below.

   ```
   c:\gsc\sio4.rtx\txrate\mvs10\txrate.sln
   ```

2. Select the desired application configuration to be built. Do so with the Configuration Manager, which is accessed with the Visual Studio menu selection "Build | Configuration Manager…" In the "Configuration Manager" window that appears, under the "Active solution configuration:" list, select either the "Debug" or the "Release" option. Then click the "Close" button.

3. Remove all existing build targets by selecting the Visual Studio menu option "Build | Clean Solution". The output in the status area should indicate one success, no failures and nothing skipped.

4. Rebuild all build targets by selecting the Visual Studio menu option "Build | Rebuild Solution". The output in the status area should indicate no errors, no warnings, one success, no failures and nothing skipped.

8.7.2. Execute

Follow the below steps to execute the sample application, which should have been built using the above procedure.

1. Using Visual Studio, the easiest way to run the program is to select the menu option "Debug | Start Debugging". Any command line options can be set in the project's Property Pages.

2. The application can also be started from a command prompt as given below. For a release build, enter "`Release`" instead "`Debug`". A single iteration should take just a few seconds. Enter command line arguments per the table below.

   ```
   cd c:\gsc\sio4.rtx\txrate\mvs10\Debug\
   .\txrate.exe
   ```

Argument	Description
`-async`	If the board is Zilog UART based, then the output rate is calculated for Asynchronous operation.
`-b#`	This is the desired or beginning bit rate or oscillator frequency.
`-best`	This applies to Zilog based SIO4 boards only, and then only when using the `-sync` argument. If the argument is specified then the application examines all necessary configurations in order to identify the configuration that best satisfies the requested bit rate. This can slow the application noticeably. If the argument is not specified, then the application chooses a configuration using the highest possible oscillator frequency.
`-e#`	This is the ending bit rate or oscillator frequency, if a range is to be calculated.
`-h#`	Hold the `-b#` rate at the cable interface for this number of seconds. This is ignored if the argument `-e#` is also specified. The default is 30 seconds.
`-m`	Measure the oscillator frequency for each scan value. This is used only if the`-osc` and `-p` arguments are also given. If the `-s` option is also given then the reported results are also saved to the file.
`-osc`	Report oscillator configuration data for the specified frequencies.

-p	Program the oscillator for each scan value. This is used only if the-osc argument is also given. If the -s option is also given then the reported results are also saved to the file.
-s	Save the scan data to the file txrate.txt.
-sync	If the board is Zilog UART model, then the output rate is calculated for HDLC operation. If the board is a SYNC model, then the output is calculated for –SYNC boards. This is the default.
index	This is the index of the channel to access.

NOTE: If the -b and the -e options are both given, then the operation is carried out for each bit rate in the specified inclusive range.

NOTE: If the -b option is given and the -e option is omitted, then the results computed for the specified -b bit rate will appear at the cable interface for the -h specified period.

NOTE: If the -osc option is given and the -b and -e options are omitted, then the oscillator output will appear at the cable interface.

8.8. ASYNC: asyncc2c – Asynchronous Channel-to-Channel Data Transfer

This sample console application uses the Asynchronous serial protocol for data transfer between designated transmit and receive channels. The channels designated may be the same exact channel, two different channels on the same board, or two channels on two different boards.

8.8.1. Build

Follow the below steps to build the sample application.

1. Load the Visual Studio solutions file listed below.

   ```
   c:\gsc\sio4.rtx\async\asyncc2c\mvs10\asyncc2c.sln
   ```

2. Select the desired application configuration to be built. Do so with the Configuration Manager, which is accessed with the Visual Studio menu selection "Build | Configuration Manager…" In the "Configuration Manager" window that appears, under the "Active solution configuration:" list, select either the "Debug" or the "Release" option. Then click the "Close" button.

3. Remove all existing build targets by selecting the Visual Studio menu option "Build | Clean Solution". The output in the status area should indicate one success, no failures and nothing skipped.

4. Rebuild all build targets by selecting the Visual Studio menu option "Build | Rebuild Solution". The output in the status area should indicate no errors, no warnings, one success, no failures and nothing skipped.

8.8.2. Execute

Follow the below steps to execute and exercise the test application.

> **NOTE**: When the specified transmit and receive channel indexes are different, a strait passthrough cable is required to properly connect the two channels to one another. The result is that the cabling directly connects all transmit channel pins to their identical receiver channel pins.

1. Using Visual Studio, the easiest way to run the program is to select the menu option "Debug | Start Debugging". Any command line options can be set in the project's Property Pages.

2. The application can also be started from a command prompt as given below. For a release build, enter "`Release`" instead "`Debug`". The application will perform a series of tests followed by a period of data transfer. A single iteration should take a little over 30 seconds. Enter command line arguments per the table below.

   ```
   cd c:\gsc\sio4.rtx\async\asyncc2c\mvs10\Debug\
   .\asyncc2c.exe
   ```

Argument	Description
`-c`	Perform continuous testing until an error occurs.
`-C`	Perform continuous testing, continuing even if errors occur.
`-m#`	Limit continuous testing to this many minutes.
`-n#`	Limit continuous testing to this number of iterations.
`tx#`	This is the zero based index of the transmit channel.
`rx#`	This is the zero based index of the receive channel.

8.9. ISOC: isocc2c – Isochronous Channel-to-Channel Data Transfer

This sample console application uses the Isochronous serial protocol for data transfer between designated transmit and receive channels. The channels designated may be the same exact channel, two different channels on the same board, or two channels on two different boards.

8.9.1. Build

Follow the below steps to build the sample application.

1. Load the Visual Studio solutions file listed below.

   ```
   c:\gsc\sio4.rtx\isoc\isocc2c\mvs10\isocc2c.sln
   ```

2. Select the desired application configuration to be built. Do so with the Configuration Manager, which is accessed with the Visual Studio menu selection "Build | Configuration Manager…" In the "Configuration Manager" window that appears, under the "Active solution configuration:" list, select either the "Debug" or the "Release" option. Then click the "Close" button.

3. Remove all existing build targets by selecting the Visual Studio menu option "Build | Clean Solution". The output in the status area should indicate one success, no failures and nothing skipped.

4. Rebuild all build targets by selecting the Visual Studio menu option "Build | Rebuild Solution". The output in the status area should indicate no errors, no warnings, one success, no failures and nothing skipped.

8.9.2. Execute

Follow the below steps to execute and exercise the test application.

> **NOTE**: When the specified transmit and receive channel indexes are different, a strait passthrough cable is required to properly connect the two channels to one another. The result is that the cabling directly connects all transmit channel pins to their identical receiver channel pins.

1. Using Visual Studio, the easiest way to run the program is to select the menu option "Debug | Start Debugging". Any command line options can be set in the project's Property Pages.

2. The application can also be started from a command prompt as given below. For a release build, enter "`Release`" instead "`Debug`". The application will perform a series of tests followed by a period of data transfer. A single iteration should take a little over 30 seconds. Enter command line arguments per the table below.

   ```
   cd c:\gsc\sio4.rtx\isoc\isocc2c\mvs10\Debug\
   .\isocc2c.exe
   ```

Argument	Description
`-c`	Perform continuous testing until an error occurs.
`-C`	Perform continuous testing, continuing even if errors occur.
`-m#`	Limit continuous testing to this many minutes.
`-n#`	Limit continuous testing to this number of iterations.
`tx#`	This is the zero based index of the transmit channel.
`rx#`	This is the zero based index of the receive channel.

8.10. SYNC: gpio – General Purpose I/O

This sample console application performs a series of tests using the –SYNC model board's GPIO cable interface capabilities.

8.10.1. Build

Follow the below steps to build/rebuild the sample application.

1. Load the Visual Studio solutions file listed below.

   ```
   c:\gsc\sio4.rtx\sync\gpio\gpio.sln
   ```

2. Select the desired application configuration to be built. Do so with the Configuration Manager, which is accessed with the Visual Studio menu selection "Build | Configuration Manager…" In the "Configuration Manager" window that appears, under the "Active solution configuration:" list, select either the "Debug" or the "Release" option. Then click the "Close" button.

3. Remove all existing build targets by selecting the Visual Studio menu option "Build | Clean Solution". The output in the status area should indicate one success, no failures and nothing skipped.

4. Rebuild all build targets by selecting the Visual Studio menu option "Build | Rebuild Solution". The output in the status area should indicate no errors, no warnings, one success, no failures and nothing skipped.

8.10.2. Execute

Follow the below steps to execute the sample application, which should have been built using the above procedure.

1. Using Visual Studio, the easiest way to run the program is to select the menu option "Debug | Start Debugging". Any command line options can be set in the project's Property Pages.

2. The application can also be started from a command prompt as given below. For a release build, enter "`Release`" instead "`Debug`". The application will perform a series of tests followed by a period of data transfer. A single iteration should take a little over 30 seconds. Enter command line arguments per the table below.

   ```
   cd c:\gsc\sio4.rtx\sync\gpio\mvs10\Debug\
   .\gpio.exe
   ```

Argument	Description
`-232`	Select RS-232 cable transceivers.
`-422`	Select RS-422/RS-485 cable transceivers.
`-423`	Select RS-423 cable transceivers.
`-c`	Perform continuous testing until an error occurs.
`-C`	Perform continuous testing, continuing even if errors occur.
`-m#`	Limit continuous testing to this many minutes.
`-n#`	Limit continuous testing to this number of iterations.
`-s#`	Hold output and monitor input for this long (seconds).
`index`	The zero based index of the SIO4 channel to access.

8.11. SYNC: syncc2c – SYNC Channel-to-Channel Data Transfer

This sample console application uses the Synchronous serial protocol for data transfer between designated transmit and receive channels. The channels designated may be the same exact channel, two different channels on the same board, or two channels on two different boards.

8.11.1. Build

Follow the below steps to build/rebuild the sample application.

1. Load the Visual Studio solutions file listed below.

   ```
   c:\gsc\sio4.rtx\sync\syncc2c\syncc2c.sln
   ```

2. Select the desired application configuration to be built. Do so with the Configuration Manager, which is accessed with the Visual Studio menu selection "Build | Configuration Manager…" In the "Configuration Manager" window that appears, under the "Active solution configuration:" list, select either the "Debug" or the "Release" option. Then click the "Close" button.

3. Remove all existing build targets by selecting the Visual Studio menu option "Build | Clean Solution". The output in the status area should indicate one success, no failures and nothing skipped.

4. Rebuild all build targets by selecting the Visual Studio menu option "Build | Rebuild Solution". The output in the status area should indicate no errors, no warnings, one success, no failures and nothing skipped.

8.11.2. Execute

Follow the below steps to execute the sample application, which should have been built using the above procedure.

1. Using Visual Studio, the easiest way to run the program is to select the menu option "Debug | Start Debugging". Any command line options can be set in the project's Property Pages.

2. The application can also be started from a command prompt as given below. For a release build, enter "`Release`" instead "`Debug`". The application will perform a series of tests followed by a period of data transfer. A single iteration should take a little over 30 seconds. Enter command line arguments per the table below.

   ```
   cd c:\gsc\sio4.rtx\sync\syncc2c\mvs10\Debug\
   .\syncc2c.exe
   ```

Argument	Description
`-c`	Perform continuous testing until an error occurs.
`-C`	Perform continuous testing, continuing even if errors occur.
`-m#`	Limit continuous testing to this many minutes.
`-n#`	Limit continuous testing to this number of iterations.
`tx#`	This is the zero based index of the transmit channel.
`rx#`	This is the zero based index of the receive channel.

9. 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 and makes no attempt at explaining configuration of the Zilog Z16C30. This is simply to address a very few issues relating to GSC specific features of the SIO4.

9.1. Debugging Aid

The driver archive includes numerous utility services. Of specific interest is the function `sio4_reg_list()`, which provides a detailed register dump of a channel's registers. This is especially useful for customer support when generated at a point in your code just prior to the appearance of a problem. The prototype for this service is in the utility header `sio4_utils.h`. The service itself is included in the library `sio4_utils.lib`. The header is in the `sio4.rtx\include` directory while the library is located in the directory `sio4.rtx\utils`.

9.2. SIO4 Zilog Configuration Aid

The figure below is intended as an aid to those trying to configure an SIO4 with a USC.

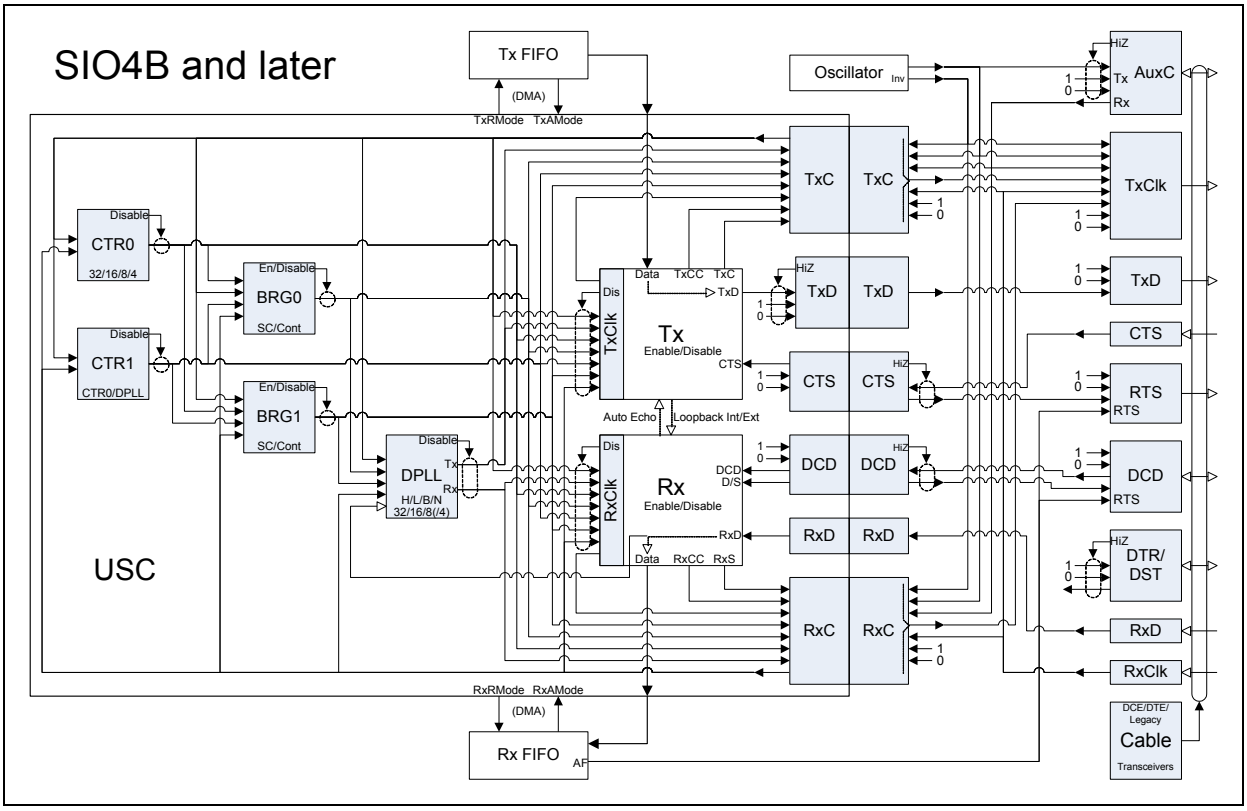

Figure 2 A configuration aid for Zilog based SIO4B and later boards.

9.3. SIO4-SYNC Configuration Aid

The figure below is intended as an aid to those trying to configure SIO4-SYNC model boards.

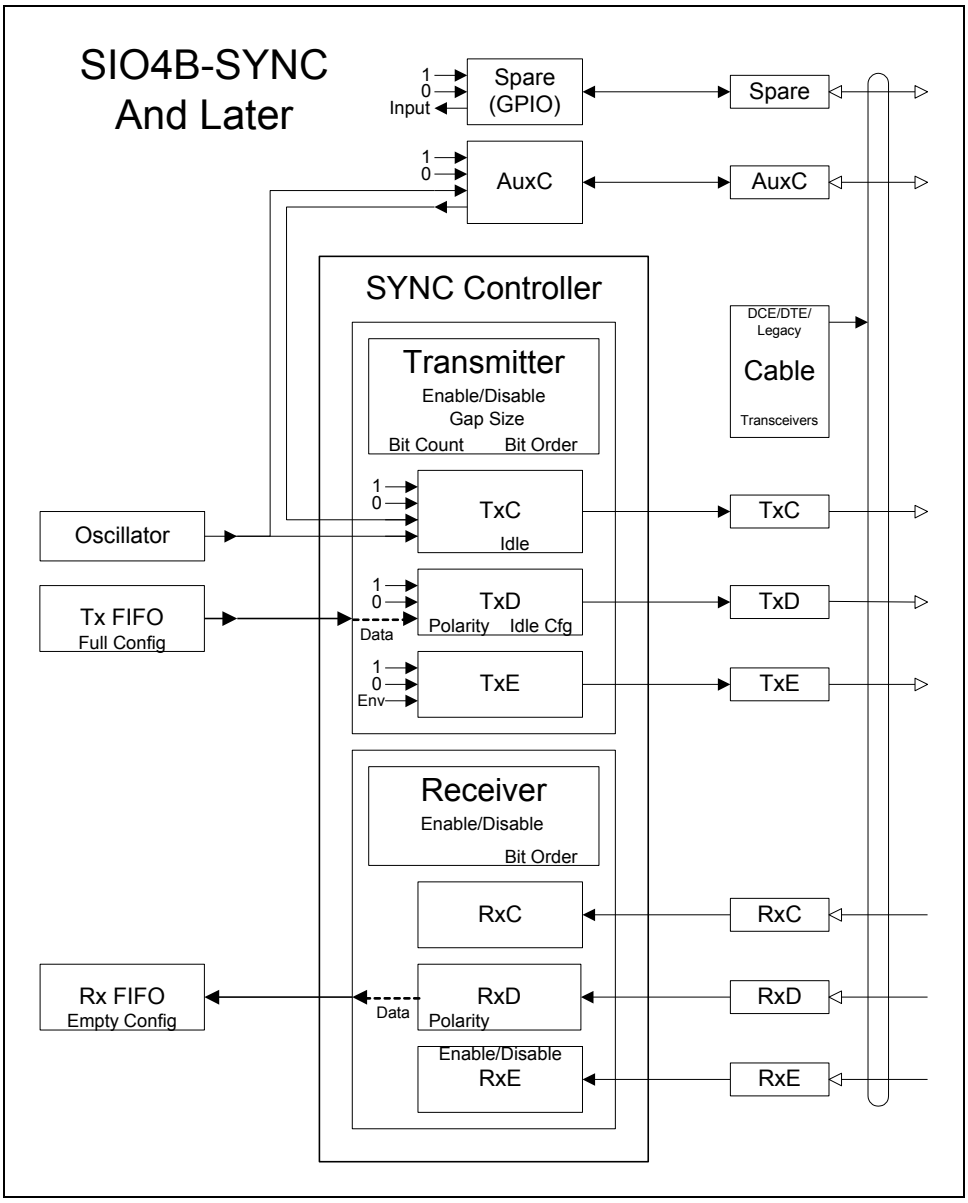

Figure 3 A configuration aid for SIO4B and later –SYNC boards.

Document History

Revision	Description
November 5, 2015	Initial release of the RTX Device Driver. Version 3.1.64.0.0.