General Standards Corporation
SIO4 Application Interface
User Manual version 1.6.4

September 4, 2013

Table of Contents

Introduction

This document describes the Application Programmers Interface (API) for the General Standards Corporation I/O Interface boards. Some API functions apply only to certain hardware. Each function contains the list of boards that it supports. For examples of how to use the API functions, refer to the source code included in the API examples. These examples are located in the sub-directories named samples/SIO4B_Test on Win32 systems and in /usr/local/GscApi/Examples on linux systems.

This API was written using Microsoft Visual Studio .NET 2003 and is compatible with C#.Net and VB.Net as well as Win32 console and MFC applications, both "managed" and "unmanaged". Microsoft Visual C++ 6.0 is supported as well. The API also supports the Linux platform and the GNU C compiler.

Win32 Installation

The API support files are installed during the standard installation of the driver. The API support files are placed, by default, into the C:\Program Files\General Standards Corporation\GscApi\ directory and subdirectories and consist of the following files:

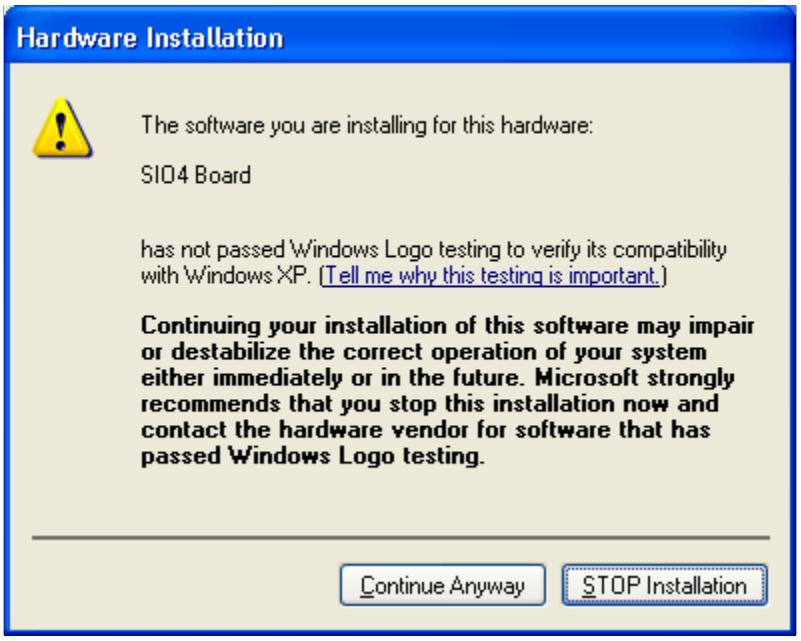
GscApi.h – This is the header file that should be included in any source files that utilize the API. This file contains the function prototypes and constant definitions needed to access the API.

GscApi.lib – This is the import library file that should be included in your project so that the linker can find the API functions.

GscApi.dll – This is the dynamically linked library file that contains the actual API code. It should be located in the same directory as your executable or in your system path so that your application can access the API functions. This file will also be installed to your system32 directory during installation.

It is recommended that you install the driver/API before installing the SIO4B card. After the installation completes, shut the system down and install the SIO4B card.

Under Windows XP, you may get the following warning during the Hardware Wizard's installation of the card. You can safely choose Continue Anyway to install the driver.



Linux Installation

On the linux platform, the General Standards Corp API support files are packaged in an autotools tarball file called GscApi-1.3.2.tar.gz. To install the API support files, follow the standard installation process:

1. Copy the tarball file into a directory where the files can be extracted. Change directories to the newly created directory. Extract the files in the tar archive with the command tar xzvf GscApi-1.3.2.tar.gz. This will create a subdirectory called GscApi-1.3.2 containing the installation files. One of the files in this directory is the INSTALL file, which also contains instructions for installing the GSC API.
2. Without changing directories, create a new directory named buildGsc, from which the installation process will be run. The linux command to do this is mkdir buildGsc.
3. Change directories to the buildGsc directory and run the following command: ../GscApi-1.3.2/configure. This will run a series of checks of your linux platform to make sure the libraries and header files needed by the API are present. If the configure command fails some of its checks, install the missing software and rerun the configure command. See the notes at the end of this section for instructions regarding how to install software that may be missing from your linux distribution.
4. After the configure command has been executed successfully, type "make" from the command line. This will build the Gsc API library and samples.

To install the newly built API files, type "make install" from the command line. This completes the installation process. The API support files are placed, by default, into the /usr/local/include/GscApi and /usr/local/lib directories and consist of the following files:

GscApi.h – This is the header file that should be included in any source files that utilize the API. This file contains the function prototypes and constant definitions needed to access the API.

libGscApi – This is the shared library that contains the actual API code. It should be located in the same directory as your executable or in your system path so that your application can access the API functions. This file will also be installed to your system32 directory during installation.

The driver source is installed in the /usr/local/GscApi/PlxLinux directory tree. The driver must be manually built and loaded as a module. Currently, the Plx Linux driver is not available on kernel.org, so it is not built into any distributed linux kernels. To build the driver source, follow these steps:

1. Make sure the environment variable PLX_SDK_DIR is defined and exported. This should be done by adding the following line to the .profile file in the user's home directory:

   export PLX_SDK_DIR=/usr/local/GscApi/PlxLinux.

2. To build the Pci9080 driver, type the following from a shell prompt:

   cd $PLX_SDK_DIR/driver
   ./buildriver 9080.

3. To install the driver, type the following:

   cd $PLX_SDK_DIR/bin
   ./modload 9080

Once the driver is installed, any of the Gsc or Plx sample applications can be executed. All the sample applications are built during installation and then installed in the usr/local/bin directory. This directory should be in the PATH environment variable on your linux system, so the samples can be executed from any directory.

The source code of each sample application may also be built with the supplied makefiles. However, one more step is required before attempting to build any of the sample code. In order to use the GscApi and PciApi shared libraries, the file /etc/ld.so.conf must include the following line:

/usr/local/lib

If the file does not contain this line, edit the file and add the line. The next time the linux system is booted, the linux dynamic linker run-time bindings will be updated to

include the Gsc and Plx libraries. The command 'ldconfig' may be used to update the linker run-time bindings if a system reboot is undesirable.

To build a sample application, change directories to the desired application and type 'make' from a shell prompt. The sample applications are located in /usr/local/GscApi/Examples. The resulting binary executable is written to the 'App' subdirectory of the sample source directory. For example, to build and run the DisplayBoards sample application, type the following from a shell command prompt:

```
cd /usr/local/GscApi/Examples/DisplayBoards
make
cd App
./DisplayBoards
```

Note that there is an environment variable called PLX_DEBUG that is recognized by the sample application makefiles. This variable may be defined in the sample application makefiles by uncommenting the following line in the desired makefile:

```
#PLX_DEBUG          = 1
```

If this variable is defined, then a debug executable will be built, with the text "_dbg" suffixed to the filename. In the example above, if the DisplayBoards application is built for debugging, the name of the executable generated would be DisplayBoards_dbg. It will be written to the 'App' subdirectory, just as it is for the non-debug version of the application.

The API installation also includes the source code for the GscApi library. It is located in /usr/local/GscApi/src. The library can be built from the source code with the makefile provided along with the source code. To build the GscApi library, change directories to /usr/local/GscApi/src and type "make" from the command line. This will build a static library and place it in /usr/local/GscApi/src/Library directory. To link this library with applications instead of the provided shared library, modify the application makefile as appropriate.

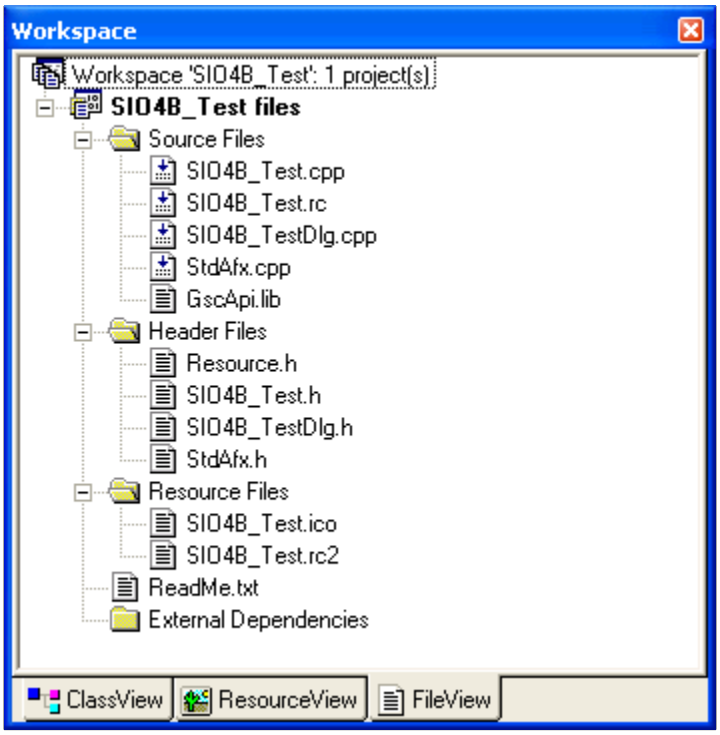
Win32 Project Setup

To utilize the SIO4B-API in your software application, you should include the GscApi.h header file and the GscApi.lib static library file in your project. The details of adding these files to your project will differ from compiler to compiler. We will concentrate on the Microsoft® Visual Studio .NET 2003 IDE. Support for other compilers may be added in future releases of the SIO4B-API.

First, be sure that your source code file that will make use of the SIO4B-API has the GscApi.h header file included as follows:

```
#include "GscApi.h"
```

Next, be sure that your project has the GscApi.lib static library file included to be compiled as part of your project as follows (here is a sample of the workspace for the SIO4B_Test included with the SIO4B-API in the Samples directory):



Lastly, the GscApi.dll file should be made available to your final executable program – either by having this file in the same directory or making it available via your system's path.

Linux Project Setup

The standard GNU compiler is supported on Linux. To utilize the SIO4B-API in your software application, you should include the GscApi.h header file in any source code files that reference API functions or data types just as you would .

The only other requirement for writing application code to use the API is to add the GscApi and PlxApi libraries to the GNU linker in your makefile. A makefile that builds a sample application called MyApp, consisting of one c source file called MyApp.c, would contain linker-related script that looks like the following:

```
# definition of linker
LINK = libtool –mode=link $(LDFLAGS) –o $@

# definition of linker flags – here is where the libraries are added to the build.
LDFLAGS =
LIBS = -lGscApi –lPlxApi

# suffix rule to invoke linker
MyApp : MyApp.o $(DEPENDENCIES)
    $(LINK) $(LDFLAGS) MyApp.o $(LIBS)
```

System Level Routines

The System Level Routines perform functions that either apply to all SIO4 boards in the system, or are not board specific. These routines are used to gather information about the current system setup. All of these functions return zero if successful or a non-zero error code if a failure occurs.

GscFindBoards

GscFindBoards(…) is used to report the number of GSC SIO4 boards in the system as well as some board specific information. An application may call this function at any time.

Supported Hardware:

All

Prototype:

```
int GscFindBoards(
                        int *boardCount,
                        GSC_DEVICES_STRUCT *results);
```

Parameters:

boardCount – a pointer to the location to save the number of boards detected. This value will be zero if no boards are found.

results – a pointer to the devices structure that will be filled in with the information from the boards found. If this parameter is NULL, no board specific information will be returned. The boardCount will, however, still be returned. The devices structure is defined as follows:

```
typedef struct
{
        int     busNumber;          // Identifies the bus that contains the board
        int     slotNumber;         // Identifies the slot that contains the board
        int     vendorId;           // Identifies the board Vendor
        int     deviceId;           // Identifies the device
        char    serialNumber[25];   // A unique board serial number
} GSC_DEVICES_STRUCT;
```

GscGetErrorString

GscGetErrorString(…) is used to translate the error codes that are returned by the various API functions into meaningful null-terminated strings. The strings returned by this function are guaranteed to be less than 80 characters in length.

Supported Hardware:

All

Prototype:

```
int GscGetErrorString(
                                int errorCode,
                                char *errorString);
```

Parameters:

errorCode – the error code returned by an API function.

errorString – a pointer to a character string that will be filled with the text that corresponds to the errorCode.

Board Level Routines

The Board Level Routines perform functions that apply to a single SIO4 board. These functions affect all channels of the SIO4 board. Each of these routines requires the board number (*boardNumber*) as the first argument. The board numbers run from 1 up to the number that is returned from the call to GscFindBoards(…) function. Note that this number will always be 1 in a single board system.

These routines can be called at any time. All of these functions return zero if successful or a non-zero error code if a failure occurs.

GscOpen

GscOpen(…) is used to "open" the SIO4 board for operation. It should be called before any other Board or Channel Level routines and should only be called once. In the process of opening a board, all four channels are reset and the clock outputs are disabled.

Supported Hardware:

All

Prototype:

```
int GscOpen( int boardNumber,    int headerVersion);
```

Parameters:

boardNumber – The number of the desired board. This number corresponds to the results of the GscFindBoards(…) function. Note that this number will always be 1 in a single board system.

headerVersion – The version of the api being used by the application. The value GSC_API_VERSION, from the GscApi.h header file, should always be passed in for this parameter.

GscClose

GscClose(…) is used to "close" the SIO4 board. It should be the last API function called before the application terminates. This function releases the resources that are used by the API and driver.

Supported Hardware:

All

Prototype:

int GscClose(int boardNumber);

Parameters:

boardNumber – The number of the desired board. This number corresponds to the results of the GscFindBoards(…) function. Note that this number will always be 1 in a single board system.

GscGetInfo

GscGetInfo(…) returns general information about an SIO4 board. The information is returned in a board info structure.

Supported Hardware:

All

Prototype:

```
int GscGetInfo(
                        int boardNumber,
                        PBOARD_INFO info);
```

Parameters:

boardNumber – The number of the desired board. This number corresponds to the results of the GscFindBoards(…) function. Note that this number will always be 1 in a single board system.

PBOARD_INFO info – a pointer to the BOARD_INFO structure that holds the retrieved board information. The BOARD_INFO structure is defined as follows:

```
typedef struct
{
        char    apiVersion[20];         // The installed GscApi library version
        char    driverVersion[20];      // The installed plx driver version
        char    fpgaVersion[20];        // The fpga version
        char    boardType[50];          // The board type, retrieved from the fpga.
} BOARD_INFO, *PBOARD_INFO;
```

GscGetVersions

GscGetVersions(…) returns the various version numbers associated with the API, the low level driver, and the SIO4 board's FPGA. The Library and Driver version numbers are returned in the form: 0xMMmmee where MM is the major release number, mm is the minor release number, and ee is the engineering release number. The entire version is defined as MM.mm.ee for example 1.02.05 is returned as 0x00010205. The FPGA version number has several encoded fields. The low byte contains the actual version number. Refer to the hardware users manual for details on the other encoded fields.

Supported Hardware:

All

Prototype:

```
int GscGetVersions(
                        int boardNumber,
                        int *libVersion,
                        int *driverVersion,
                        int *fpgaVersion);
```

Parameters:

boardNumber – The number of the desired board. This number corresponds to the results of the GscFindBoards(…) function. Note that this number will always be 1 in a single board system.

libVersion – A pointer to the location that will receive the library (API) version number. If this value is NULL, no value will be returned.

driverVersion – A pointer to the location that will receive the low level driver version number. If this value is NULL, no value will be returned.

fpgaVersion – A pointer to the location that will receive the FPGA firmware version number. If this value is NULL, no value will be returned.

GscLocalRegisterRead

GscLocalRegisterRead(…) is used to read the local board registers. These registers reside within the board's FPGA. It is not recommended that a user application directly access these registers. This function is included for diagnostic purposes only.

Supported Hardware:

All

Prototype:

```
int GscLocalRegisterRead(
                                int boardNumber,
                                int reg,
                                int *value);
```

Parameters:

boardNumber – The number of the desired board. This number corresponds to the results of the GscFindBoards(…) function. Note that this number will always be 1 in a single board system.

reg – The address of the register to be read. Macros for these addresses are described in the section titled "Local Register Definitions".

value – A pointer to the location that will receive the results of the read operation.

GscLocalRegisterWrite

GscLocalRegisterWrite(…) is used to write to the local board registers. These registers reside within the board's FPGA. It is not recommended that a user application directly access these registers. This function is included for diagnostic purposes only.

Supported Hardware:

All

Prototype:

```
int GscLocalRegisterWrite(
                                    int boardNumber,
                                    int reg,
                                    int value);
```

Parameters:

boardNumber – The number of the desired board. This number corresponds to the results of the GscFindBoards(…) function. Note that this number will always be 1 in a single board system.

reg – The address of the register to be written. Macros for these addresses are described in the section titled "Local Register Definitions".

value – The value that is to be written to the local register.

GscAllocPhysicalMemory

GscAllocPhysicalMemory(…) is used to attempt to allocate a physically contiguous, page-locked buffer which is safe for use with DMA operations.

Supported Hardware:

All

Prototype:

```
int GscAllocPhysicalMemory(
                                    int boardNumber,
                                    PLX_PHYSICAL_MEM *pciMem);
```

Parameters:

boardNumber – The number of the desired board. This number corresponds to the results of the GscFindBoards(…) function. Note that this number will always be 1 in a single board system.

pcimem – A pointer to a PLX_PHYSICAL_MEM structure that will contain the buffer information.

GscMapPhysicalMemory

GscMapPhysicalMemory(…) is used to map into user virtual space a buffer previously allocated with GscAllocPhysicalMemory.

Supported Hardware:

All

Prototype:

```
int GscMapPhysicalMemory(
                                    int boardNumber,
                                    PLX_PHYSICAL_MEM *pciMem);
```

Parameters:

boardNumber – The number of the desired board. This number corresponds to the results of the GscFindBoards(…) function. Note that this number will always be 1 in a single board system.

pcimem – A pointer to a PLX_PHYSICAL_MEM structure that will contain the buffer information.

GscUnmapPhysicalMemory

GscUnmapPhysicalMemory(…) is used to unmap a buffer previously mapped into user virtual space with GscAllocPhysicalMemory.

Supported Hardware:

All

Prototype:

```
int GscUnmapPhysicalMemory(
                                    int boardNumber,
                                    PLX_PHYSICAL_MEM *pciMem);
```

Parameters:

boardNumber – The number of the desired board. This number corresponds to the results of the GscFindBoards(…) function. Note that this number will always be 1 in a single board system.

pcimem – A pointer to a PLX_PHYSICAL_MEM structure that will contain the buffer information.

Channel Level Routines

The Channel Level Routines perform functions that apply to a single channel on an SIO4 board. Each of these routines requires the board number (boardNumber) as the first parameter and the channel number (channel) as the second parameter. The board number corresponds to the results of the GscFindBoards(…) function. Note that this number will always be 1 in a single board system. The channel number will always be 1, 2, 3, or 4.

These routines can be called at any time. All of these functions return zero if successful or a non-zero error code if a failure occurs.

GscSio4ChannelReset

GscSio4ChannelReset(…) resets a single channel on the SIO4 board. In addition to disabling the serial channel, this function sets the "Almost Empty" and "Almost Full" FIFO flags to 16.

Supported Hardware:

All

Prototype:

```
int GscSio4ChannelReset(
                        int boardNumber,
                        int channel);
```

Parameters:

boardNumber – The number of the desired board. This number corresponds to the results of the GscFindBoards(…) function. Note that this number will always be 1 in a single board system.

channel – The desired channel number. This number will be 1, 2, 3, or 4.

GscSio4ChannelResetRxFifo

GscSio4ChannelResetRxFifo(…) resets the Rx FIFO for a single channel. After the reset, the FIFO will contain no data.

Supported Hardware:

All

Prototype:

```
int GscSio4ChannelResetRxFifo (
                                        int boardNumber,
                                        int channel);
```

Parameters:

boardNumber – The number of the desired board. This number corresponds to the results of the GscFindBoards(…) function. Note that this number will always be 1 in a single board system.

channel – The desired channel number. This number will be 1, 2, 3, or 4.

GscSio4ChannelResetTxFifo

GscSio4ChannelResetTxFifo(…) resets the Tx FIFO for a single channel. After the reset, the FIFO will contain no data.

Supported Hardware:

All

Prototype:

```
int GscSio4ChannelResetTxFifo (
                                        int boardNumber,
                                        int channel);
```

Parameters:

boardNumber – The number of the desired board. This number corresponds to the results of the GscFindBoards(…) function. Note that this number will always be 1 in a single board system.

channel – The desired channel number. This number will be 1, 2, 3, or 4.

GscSio4ChannelRegisterRead

GscSio4ChannelRegisterRead(…) is used to read the registers in the Universal Serial Chip that controls the specified channel. It is not recommended that a user application directly access these registers. This function is included for diagnostic purposes only.

Supported Hardware:

PCI-SIO4B

Prototype:

```
int GscSio4ChannelRegisterRead(
                                    int boardNumber,
                                    int channel,
                                    int reg,
                                    int *value);
```

Parameters:

boardNumber – The number of the desired board. This number corresponds to the results of the GscFindBoards(…) function. Note that this number will always be 1 in a single board system.

channel – The desired channel number. This number will be 1, 2, 3, or 4.

reg – The address of the register to be read. Macros for these addresses are described in the section titled “Channel Register Definitions”.

value – A pointer to the location that will receive the results of the read operation.

GscSio4ChannelRegisterWrite

GscSio4ChannelRegisterWrite(…) is used to write to the registers in the Universal Serial Chip that controls the specified channel. It is not recommended that a user application directly access these registers. This function is included for diagnostic purposes only.

Supported Hardware:

PCI-SIO4B

Prototype:

```
int GscSio4ChannelRegisterWrite(
                                    int boardNumber,
                                    int channel,
                                    int reg,
                                    int value);
```

Parameters:

boardNumber – The number of the desired board. This number corresponds to the results of the GscFindBoards(…) function. Note that this number will always be 1 in a single board system.

channel – The desired channel number. This number will be 1, 2, 3, or 4.

reg – The address of the register to be written. Macros for these addresses are described in the section titled "Channel Register Definitions".

value – The value that is to be written to the register.

GscSio4GetLastError

GscSio4GetLastError(…) is used to retrieve the error description text of the last channel-level api call made for the specified channel.

Supported Hardware:

All

Prototype:

```
int GscSio4GetLastError(
                                    int boardNumber,
                                    int channel,
                                    int errorCode,
                                    char *errorString
                                    char *errorDetail);
```

Parameters:

boardNumber – The number of the desired board. This number corresponds to the results of the GscFindBoards(…) function. Note that this number will always be 1 in a single board system.

channel – The desired channel number. This number will be 1, 2, 3, or 4.

errorCode – The integer error code

errorString – The error description text

errorDetail – More verbose and detailed error description text

GscSio4ChannelSetMode / GscSio4ChannelGetMode

GscSio4ChannelSetMode(…) sets a single channel of the SIO4 board to the desired serial format and bit rate.

Each mode has its own defaults, as described below, which can be altered by calling the appropriate Channel Level Routines after this function returns.

Supported Hardware:

All

Prototype:

```
int GscSio4ChannelSetMode(
                                int boardNumber,
                                int channel,
                                int mode,
                                int bitRate);

int GscSio4ChannelGetMode(
                                int boardNumber,
                                int channel,
                                int *mode,
                                int *bitRate);
```

Parameters:

boardNumber – The number of the desired board. This number corresponds to the results of the GscFindBoards(…) function. Note that this number will always be 1 in a single board system.

channel – The desired channel number. This number will be 1, 2, 3, or 4.

mode – The desired/current serial mode for this channel. The value should be one of the following:

GSC_MODE_ASYNC – Sets the channel to standard asynchronous mode. The channel defaults to 8 data bits, no Parity, and one stop bit. It also uses a 16x sampling clock.

GSC_MODE_ISO – Sets the channel to isochronous mode. Uses the same defaults as GSC_MODE_ASYNC except the sampling clock, which is set to 1x.

GSC_MODE_HDLC – Sets the channel to HDLC mode. The Transmit clock is derived from the on-board source at the rate specified (*bitRate*) and is also driven onto the cable for use by the receiving end. The receiver clock is connected to the cable and should be supplied by the transmitter at the other end.

GSC_MODE_SYNC -
GSC_MODE_SYNC_ENV – (SIO4-SYNC boards only)
GSC_MODE_ASYNC_CV -
GSC_MODE_MONOSYNC -
GSC_MODE_BISYNC -
GSC_MODE_TRANS_BISYNC –

GSC_MODE_NBIF -
GSC_MODE_802_3 -

bitRate – The desired/current serial bit (baud) rate for this channel. This value can range from 250 to 10,000,000 for synchronous modes and 50 to 1,000,000 for asynchronous modes.

GscSio4GetOption/GscSio4SetOption

GscSio4SetOption(…) sets the value of a protocol configuration option for a channel. The available options are defined by the GSC_OPTION_NAME enumerated type.

Supported Hardware:

All

Prototype:

```
int GscSio4SetOption(
                        int boardNumber,
                        int channel,
                        enum GSC_OPTION_NAME option,
                        int value);
int GscSio4GetOption(
                        int boardNumber,
                        int channel,
                        enum GSC_OPTION_NAME option,
                        int value[]);
```

Parameters:

boardNumber – The number of the desired board. This number corresponds to the results of the GscFindBoards(…) function. Note that this number will always be 1 in a single board system.

channel – The desired channel number. This number will be 1, 2, 3, or 4.

option – The protocol option to set or retrieve. The available options are defined in the GSC_OPTION_NAME enumerated type. They are listed in the table below.

value – The value or values set or retrieved . When calling GscSio4SetOption, in some cases *value* will actually contain a pair of 16 bit values, such as when configuring the GSC_SIO_PROTOCOL option. In this case *value* will contain a protocol option in the upper 16 bits and a termination option in the lower 16 bits. When retrieving the value of this option using GscSio4GetOption, the protocol and termination options will be returned as two elements in the value[] array. The majority of the available options are represented by a single value. The options that are represented as a pair of values are listed below:

The table below lists the available options and valid settings for each option. The GscSio4SetOption() function requires a 32-bit parameter value for all options. For some of the options, this parameter value represents two 16-bit option settings rather than one 32-bit setting. For these options, the table includes descriptions of the format each 16-bit

parameter and its valid values. Likewise, for the GscSio4GetOption() function, which returns two array entries in the case of the options composed of two 16-bit values, the table describes each entry returned in the parameter array along with its set of valid values.

Option Name	***Description***	***Set Value Parameter Format***	***Get Value Parameter Format***	***Valid Values***
GSC_SIO_DATASIZE	The size of the transmitted and received data for a single channel of the SIO4 board.	[31..0]: datasize	*value*[0]: datasize	1..8 for standard SIO4 boards. 0.. 65535 for -SYNC boards
GSC_SIO_GAPSIZE	The size of the gap between transmitted data words for a single channel of the SIO4 board. The gap size can be set to any value between 0 and 65535.	[31..0]: gapsize	*value*[0]: gapsize	0..65535 for -SYNC boards only.
GSC_SIO_MSBLSBORDER	The byte ordering of both transmitted and received data words for a single channel of the SIO4 board. The order can be set to transmit or receive either the most significant byte first or the least significant byte first.	[31..16]: Tx Order [15..0]: Rx Order	*value*[0]: Tx Order *value*[1]: RxOrder	GSC_MSB_FIRST GSC_LSB_FIRST
GSC_SIO_PARITY	The type of parity that will be used on a single channel of the SIO4 board.	[31..0]: parity	*value*[0]: parity	GSC_PARITY_NONE GSC_PARITY_EVEN GSC_PARITY_ODD GSC_PARITY_MARK GSC_PARITY_SPACE
GSC_SIO_STOPBITS	The number of stop bits to use for a single channel of the SIO4 board.	[31..0]: stopbits	*value*[0]: stopbits	GSC_STOP_BITS_0 GSC_STOP_BITS_1 GSC_STOP_BITS_1_5 GSC_STOP_BITS_2
GSC_SIO_ENCODING	The encoding type for a single channel of the SIO4 board.	[31..0] encoding	value[0]: encoding	The macros defined in the section “Channel Encoding Definitions”.
GSC_SIO_PROTOCOL	The physical interface protocol and termination options. The protocol on the standard SIO4B card is fixed at RS422/RS485 or RS232 depending on the configuration set at the factory. Only the –BX cards allow this value to be changed.	[31..16]: protocol [15..0]: termination	*value*[0]: protocol *value*[1]: termination	Protocol: GSC_PROTOCOL_RS422_RS485 GSC_PROTOCOL_RS423 GSC_PROTOCOL_RS232 GSC_PROTOCOL_RS530_1 GSC_PROTOCOL_RS530_2 GSC_PROTOCOL_V35_1 GSC_PROTOCOL_V35_2 GSC_PROTOCOL_RS422_RS423_1 GSC_PROTOCOL_RS422_RS423_2 Termination: GSC_TERMINATION_ENABLED GSC_TERMINATION_DISABLED

Option Name	***Description***	***Set Value Parameter Format***	***Get Value Parameter Format***	***Valid Values***
GSC_SIO_DTEDCE	Sets a single channel of the SIO4 board to either DTE or DCE mode. Each channel defaults to DTE mode when it is configured. Setting this option is only necessary if DCE mode is required, or to switch back to DTE mode after a previous change to DCE mode. The pin-outs for both DTE and DCE modes are available in the Hardware Users Manual.	[31..0]: mode	*value*[0]: mode	GSC_PIN_DTE GSC_PIN_DCE
GSC_SIO_LOOPBACK	The loopback mode of a channel on the SIO4 board.	[31..0] loop mode	*value*[0]: loop mode	GSC_LOOP_NONE GSC_LOOP_EXTERNAL
GSC_SIO_RECEIVER	Used for enabling or disabling the receiver for a single channel on the SIO4 board..	[31..0]: mode	*value*[0]: mode	GSC_ENABLED GSC_DISABLED
GSC_SIO_TRANSMITTER	Used for enabling or disabling the transmitter for a single channel on the SIO4 board.	[31..0]: mode	*value*[0]: mode	GSC_ENABLED GSC_DISABLED
GSC_SIO_TXDATAPINMODE	Used to enable the TxD pin of a channel to be used for general purpose i/o.	[31..0]: mode	*value*[0]: mode	GSC_PIN_AUTO GSC_PIN_GPIO
GSC_SIO_RXDATAPINMODE	Used to enable the RxD pin of a channel to be used for general purpose i/o.	[31..0]: mode	*value*[0]: mode	GSC_PIN_AUTO GSC_PIN_GPIO
GSC_SIO_TXCLOCKPINMODE	Used to enable the TxC pin of a channel to be used for general purpose i/o.	[31..0]: mode	*value*[0]: mode	GSC_PIN_AUTO GSC_PIN_GPIO
GSC_SIO_RXCLOCKPINMODE	Used to enable the RxC pin of a channel to be used for general purpose i/o.	[31..0]: mode	*value*[0]: mode	GSC_PIN_AUTO GSC_PIN_GPIO
GSC_SIO_CTSPINMODE	Used to enable the CTS pin of a channel to be used for general purpose i/o.	[31..0]: mode	*value*[0]: mode	GSC_PIN_AUTO GSC_PIN_GPIO
GSC_SIO_RTSPINMODE	Used to enable the RTS pin of a channel to be used for general purpose i/o.	[31..0]: mode	*value*[0]: mode	GSC_PIN_AUTO GSC_PIN_GPIO
GSC_SIO_CLOCKSOURCE	Used to set the clock pin sources of the transmitter and receiver. This option provides for the transmitter and receiver to be configured with an internal or an external clock source.	[31..16]: Tx source [15..0]: Rx Source	N/A	GSC_CLOCK_INTERNAL GSC_CLOCK_EXTERNAL
GSC_SIO_CRCMODE	Used for setting the CRC generation/detection mode for a single channel. This routine is also used to set the initial value of the CRC register.	[31..16] crc mode [15..0] crc initial value	*value*[0]: crc mode *value*[1]: crc initial value	crc mode: GSC_CRC_NONE GSC_CRC_16 GSC_CRC_32 GSC_CRC_CCITT initial value: GSC_CRC_INIT_0 GSC_CRC_INIT_1

Option Name	***Description***	***Set Value Parameter Format***	***Get Value Parameter Format***	***Valid Values***
GSC_SIO_SYNCWORD	Used to set the sync word used on a channel.	[31..16]: Tx sync word [15..0]: Rx sync word	*value*[0]: Tx sync word *value*[1]: Rx sync word	Integer value between 0..65535.
GSC_SIO_TXUNDERRUN	Sets the data pattern to be transmitted under a Tx underrun condition.	[31..0]: Tx Underrun pattern	*value*[0]: Tx underrun pattern	GSC_SYN1 GSC_SYN0_SYN1 GSC_CRC_SYN1 GSC_CRC_SYN0_SYN1
GSC_SIO_TXPREAMBLE	Used to enable or disable the Tx preamble for a channel.	[31..0]: preamble state	*value*[0]: preamble state	GSC_ENABLED GSC_DISABLED
GSC_SIO_TXSHORTSYNC	Used set the Tx sync length (short or 8 bit) for a channel.	[31..0]: Tx sync length	*value*[0]: Tx sync length	GSC_ENABLED GSC_DISABLED
GSC_SIO_RXSYNCSTRIP	Set the Rx sync strip mode for a channel.	[31..0]: Rx sync strip mode	*value*[0]: Rx sync strip mode	GSC_ENABLED GSC_DISABLED
GSC_SIO_RXSHORTSYNC	Used to set the Rx sync length (short or 8 bit) for a channel.	[31..0]: Rx short sync length	*value*[0]: Rx short sync length	GSC_ENABLED GSC_DISABLED
GSC_SIO_TXPREAMBLELENGTH	Used to set the Tx preamble length for a channel	[31..0]: Tx preamble length	*value*[0]: Tx preamble length	GSC_PREAMBLE_8BITS GSC_PREAMBLE_16BITS GSC_PREAMBLE_32BITS GSC_PREAMBLE_64BITS
GSC_SIO_TXPREAMBLEPATTERN	Used to set the Tx preamble pattern for a channel.	[31..0]: Tx preamble pattern	value[0]: Tx preamble value	GSC_PREAMBLE_ALL_0 GSC_PREAMBLE_ALL_1 GSC_PREAMBLE_ALL_0_1 GSC_PREAMBLE_ALL_1_0
GSC_SIO_ORDERING	Used to set the byte and bit order used in bisync16 mode on a channel.	[31..16]: byte order [15..0] bit order	*value*[0]: byte order *value*[1]: bit order	byte and bit order: GSC_MSB_FIRST GSC_LSB_FIRST
GSC_SIO_MAXRXCOUNT	Used to set the maximum Rx count allowed	[31..0]: Max Rx Count	*value*[0]: Max Rx count	Integer value

GscSio4ChannelSetPinMode / GscSio4ChannelGetPinMode

GscSio4ChannelSetPinMode(…) configures the specified pin for general purpose I/O. The function can also set the specified pin for normal use.

Supported Hardware:

All

Prototype:

```
int GscSio4ChannelSetPinMode (
                                    int boardNumber,
                                    int channel,
                                    int pinName,
                                    int mode);

int GscSio4ChannelGetPinMode (
                                    int boardNumber,
                                    int channel,
                                    int pinName,
                                    int *mode);
```

Parameters:

boardNumber – The number of the desired board. This number corresponds to the results of the GscFindBoards(…) function. Note that this number will always be 1 in a single board system.

channel – The desired channel number. This number will be 1, 2, 3, or 4.

pinName – Identifier for the pin to be configured.

mode – The desired/current mode of operation for the specified pin. Valid values are defined in the GSC_TOKENS enumeration as follows:

GSC_PIN_AUTO
GSC_PIN_GPIO

GscSio4ChannelSetPinValue / GscSio4ChannelGetPinValue

GscSio4ChannelSetPinValue(…) sets the current value of the specified programmable PIN if it is configured as GPIO.

Supported Hardware:

All

Prototype:

```
int GscSio4ChannelSetPinValue (
                                    int boardNumber,
                                    int channel,
                                    int pinName,
                                    int value);

int GscSio4ChannelGetPinValue (
                                    int boardNumber,
                                    int channel,
                                    int pinName,
                                    int *value);
```

Parameters:

boardNumber – The number of the desired board. This number corresponds to the results of the GscFindBoards(…) function. Note that this number will always be 1 in a single board system.

channel – The desired channel number. This number will be 1, 2, 3, or 4.

pinName - Identifier for the pin to be configured.

value – The desired/current value of the specified pin. Accepted values are 0 and 1.

GscSio4ChannelFifoSizes

GscSio4ChannelFifoSizes(…) returns the size, in bytes, of the channel's Transmit and Receive FIFOs. The size of the Receive FIFO is returned in the upper 16 bits and the size of the Transmit FIFO is returned in the lower 16 bits of the result (*sizes*).

Supported Hardware:

PCI-SIO4B

Prototype:

```
int GscSio4ChannelFifoSizes(
                                    int boardNumber,
                                    int channel,
                                    int *sizes);
```

Parameters:

boardNumber – The number of the desired board. This number corresponds to the results of the GscFindBoards(…) function. Note that this number will always be 1 in a single board system.

channel – The desired channel number. This number will be 1, 2, 3, or 4.
sizes – A pointer to the location that will receive the size (in bytes) of the Receive (upper 16 bits) and the Transmit (lower 16 bits) FIFOs

GscSio4ChannelFifoCounts

GscSio4ChannelFifoCounts(…) returns the current number of bytes in the channel's Transmit and Receive FIFOs. The number of bytes in the Receive FIFO are returned in the upper 16 bits and the number of bytes in the Transmit FIFO are returned in the lower 16 bits of the result (*counts*).

Supported Hardware:

PCI-SIO4B

Prototype:

```
int GscSio4ChannelFifoCounts(
                                    int boardNumber,
                                    int channel,
                                    int *counts);
```

Parameters:

boardNumber – The number of the desired board. This number corresponds to the results of the GscFindBoards(…) function. Note that this number will always be 1 in a single board system.

channel – The desired channel number. This number will be 1, 2, 3, or 4.

counts – A pointer to the location that will receive the number of bytes currently in the Receive (upper 16 bits) and the Transmit (lower 16 bits) FIFOs.

GscSio4ChannelSetTxAlmost / GscSio4ChannelGetTxAlmost

GscSio4ChannelSetTxAlmost(…) programs the "Almost Full" and "Almost Empty" registers in the Transmit FIFO for a single channel. Once the values are programmed, the FIFO will be reset to force the change to take effect. This will also clear the contents of the FIFO, so this command should be done before any data transfers occur.

Supported Hardware:

PCI-SIO4B

Prototype:

```
int GscSio4ChannelSetTxAlmost(
                                    int boardNumber,
                                    int channel,
                                    int almostValue);
```

Parameters:

boardNumber – The number of the desired board. This number corresponds to the results of the GscFindBoards(…) function. Note that this number will always be 1 in a single board system.

channel – The desired channel number. This number will be 1, 2, 3, or 4.

almostValue – The 32bit value that will be programmed into the Transmitter FIFO's Almost Full (upper 16 bits) and Almost Empty (lower 16 bits) registers.

GscSio4ChannelSetRxAlmost / GscSio4ChannelGetRxAlmost

GscSio4ChannelSetRxAlmost(…) programs the "Almost Full" and "Almost Empty" registers in the Receive FIFO for a single channel. Once the values are programmed, the FIFO will be reset to force the change to take effect. This will also clear the contents of the FIFO, so this command should be done before any data transfers occur.

Supported Hardware:

PCI-SIO4B

Prototype:

```
int GscSio4ChannelSetRxAlmost(
                                    int boardNumber,
                                    int channel,
                                    int almostValue);
```

Parameters:

boardNumber – The number of the desired board. This number corresponds to the results of the GscFindBoards(…) function. Note that this number will always be 1 in a single board system.

channel – The desired channel number. This number will be 1, 2, 3, or 4.

almostValue – The 32bit value that will be programmed into the Receiver FIFO's Almost Full (upper 16 bits) and Almost Empty (lower 16 bits) registers.

GscSio4ChannelCheckForData

GscSio4ChannelCheckForData(…) determines whether a packet has been received on the specified channel. If a packet has been received, a dma transfer is initiated to return the data. The data received on the channel is transferred into the memory buffer pointed to by *buffer*. A number of bytes transferred is indicated by the value of *count*. This function may return before the transfer completes.

Supported Hardware:

All

Prototype:

```
int GscSio4ChannelCheckForData(
                                    int boardNumber,
                                    int channel,
                                    char *buffer,
                                    int *count);
```

Parameters:

boardNumber – The number of the desired board. This number corresponds to the results of the GscFindBoards(…) function. Note that this number will always be 1 in a single board system.

channel – The desired channel number. This number will be 1, 2, 3, or 4.

buffer – A pointer to the start of the data buffer that will receive the data. The buffer should be large enough to hold a packet of data.

count – The number of bytes transferred.

GscSio4ChannelReceivePacket

GscSio4ChannelReceivePacket(…) determines whether a packet has been received on the specified channel. If a packet has been received, a dma transfer is initiated to return the data. The data received on the channel is transferred into the memory buffer pointed to by *buffer*. A number of bytes transferred is indicated by the value of *count*. This function may return before the transfer completes.

Supported Hardware:

All

Prototype:

```
int GscSio4ChannelReceivePacket(
                                    int boardNumber,
                                    int channel,
                                    char *buffer,
                                    int *count,
                                    int *transferStatus);
```

Parameters:

boardNumber – The number of the desired board. This number corresponds to the results of the GscFindBoards(…) function. Note that this number will always be 1 in a single board system.

channel – The desired channel number. This number will be 1, 2, 3, or 4.

buffer – A pointer to the start of the data buffer that will receive the data. The buffer should be large enough to hold a packet of data.

count – The number of bytes that are to be transferred. Must be set to this value prior to making the function call.

transferStatus – Indicates the status of the transfer. The value will be non-zero if there are errors, such as a CRC error or an abort error, with the transfer. The status returned in this parameter will be a mask of any error bits in the UART RCSR register. Otherwise the value will be zero.

GscSio4ChannelReceiveData

GscSio4ChannelReceiveData(…) starts the reception of data on the specified channel. The data received on the channel is transferred into the memory buffer pointed to by *buffer*. A total of *count* bytes will be transferred. This function may return before the transfer completes. When this function returns, the value pointed to by *id* will contain a unique identifier that can be used to determine the progress of the transfer.

Supported Hardware:

All

Prototype:

```
int GscSio4ChannelReceiveData(
                                    int boardNumber,
                                    int channel,
                                    char *buffer,
                                    int count,
                                    int *id);
```

Parameters:

boardNumber – The number of the desired board. This number corresponds to the results of the GscFindBoards(…) function. Note that this number will always be 1 in a single board system.

channel – The desired channel number. This number will be 1, 2, 3, or 4.

buffer – A pointer to the start of the data buffer that will receive the data. The buffer should be at least *count* bytes long.

count – The number of bytes to transfer.

id – A pointer to the location that will hold the unique transfer identifier that is assigned to this transfer. This value can be used to determine when the transfer has completed.

GscSio4ChannelReceiveDataAndWait

GscSio4ChannelReceiveDataAndWait(…) starts the reception of data on the specified channel. The data received on the channel is transferred into the memory buffer pointed to by *buffer*. A total of *count* bytes will be transferred. This function will not return until the entire transfer has completed or the timeout period has expired. If a timeout occurs, the value in *bytesTransferred* will specify the number of bytes that were actually received. (Note that if no timeout occurs, the *bytesTransferred* value is undefined.)

Supported Hardware:

All

Prototype:

```
int GscSio4ChannelReceiveDataAndWait(
                                    int boardNumber,
                                    int channel,
                                    char *buffer,
                                    int count,
                                    int timeout,
                                    int *bytesTransferred);
```

Return value:

The function returns a zero if the packet transfer completes. Otherwise it returns a non-zero error code.

Parameters:

boardNumber – The number of the desired board. This number corresponds to the results of the GscFindBoards(…) function. Note that this number will always be 1 in a single board system.

channel – The desired channel number. This number will be 1, 2, 3, or 4.

buffer – A pointer to the start of the data buffer that will receive the data. The buffer should be at least *count* bytes long.

count – The number of bytes to transfer.

timeout – The desired timeout period (in milliseconds) for the transfer.

bytesTransferred – If a timeout occurs, this value will specify the total number of bytes that were actually received. If no timeout occurs, this value is undefined.

GscSio4ChannelReceivePlxPhysData

GscSio4ChannelReceivePlxPhysData(…) starts the reception of data on the specified channel. The data received on the channel is transferred into the physically contiguous memory buffer pointed to by *buffer*. The memory for this buffer must be allocated with the GscAllocPhysicalMemory function and mapped into user virtual space using the GscMapPhysicalMemory function. A total of *count* bytes will be transferred. This function may return before the transfer completes. When this function returns, the value pointed to by *id* will contain a unique identifier that can be used to determine the progress of the transfer.

Supported Hardware:

All

Prototype:

```
int GscSio4ChannelReceivePlxPhysData(
                                        int boardNumber,
                                        int channel,
                                        PLX_PHYSICAL_MEM *buffer,
                                        int count,
                                        int *id);
```

Parameters:

boardNumber – The number of the desired board. This number corresponds to the results of the GscFindBoards(…) function. Note that this number will always be 1 in a single board system.

channel – The desired channel number. This number will be 1, 2, 3, or 4.

buffer – A pointer to the start of the physically contiguous data buffer that will receive the data. The buffer should be at least *count* bytes long.

count – The number of bytes to transfer.

id – A pointer to the location that will hold the unique transfer identifier that is assigned to this transfer. This value can be used to determine when the transfer has completed.

GscSio4ChannelTransmitData

GscSio4ChannelTransmitData(…) starts the transmission of data on the specified channel. The data to be transmitted on the channel is transferred from the memory buffer pointed to by *buffer*. A total of *count* bytes will be transferred. This function may return before the transfer completes. When this function returns, the value pointed to by *id* will contain a unique identifier that can be used to determine the progress of the transfer.

Supported Hardware:

All

Prototype:

```
int GscSio4ChannelTransmitData(
                                    int boardNumber,
                                    int channel,
                                    char *buffer,
                                    int count,
                                    int *id);
```

Parameters:

boardNumber – The number of the desired board. This number corresponds to the results of the GscFindBoards(…) function. Note that this number will always be 1 in a single board system.

channel – The desired channel number. This number will be 1, 2, 3, or 4.

buffer – A pointer to the start of the data buffer that will be transmitted. The buffer should be at least *count* bytes long.

count – The number of bytes to transfer.

id – A pointer to the location that will hold the unique transfer identifier that is assigned to this transfer. This value can be used to determine when the transfer has completed.

GscSio4ChannelTransmitDataAndWait

GscSio4ChannelTransmitDataAndWait(…) starts the transmission of data on the specified channel. The data to be transmitted on the channel is transferred from the memory buffer pointed to by *buffer*. A total of *count* bytes will be transferred. This function will not return until the entire transfer has completed or the timeout period has expired. If a timeout occurs, the value in *bytesTransferred* will specify the number of bytes that were actually transmitted. (Note that if no timeout occurs, the *bytesTransferred* value is undefined.)

Supported Hardware:

All

Prototype:

```
int GscSio4ChannelTransmitDataAndWait(
                                        int boardNumber,
                                        int channel,
                                        char *buffer,
                                        int count,
                                        int timeout
                                        int *bytesTransferred);
```

Parameters:

boardNumber – The number of the desired board. This number corresponds to the results of the GscFindBoards(…) function. Note that this number will always be 1 in a single board system.

channel – The desired channel number. This number will be 1, 2, 3, or 4.

buffer – A pointer to the start of the data buffer that will be transmitted. The buffer should be at least *count* bytes long.

count – The number of bytes to transfer.

timeout – The desired timeout period (in milliseconds) for the transfer.

bytesTransferred – If a timeout occurs, this value will specify the total number of bytes that were actually transmitted. If no timeout occurs, this value is undefined.

GscSio4ChannelTransmitPlxPhysData

GscSio4ChannelTransmitPlxPhysData(…) starts the transmission of data on the specified channel. The data to be transmitted on the channel is transferred from the physically contiguous memory buffer pointed to by *buffer*. This buffer must be allocated using the GscAllocPhysicalMemory function and mapped to user virtual space using the GscMapPhysicalMemory function. A total of *count* bytes will be transferred. This function may return before the transfer completes. When this function returns, the value pointed to by *id* will contain a unique identifier that can be used to determine the progress of the transfer.

Supported Hardware:

All

Prototype:

```
int GscSio4ChannelTransmitPlxPhysData(
                                    int boardNumber,
                                    int channel,
                                    PLX_PHYSICAL_MEM *buffer,
                                    int count,
                                    int *id);
```

Parameters:

boardNumber – The number of the desired board. This number corresponds to the results of the GscFindBoards(…) function. Note that this number will always be 1 in a single board system.

channel – The desired channel number. This number will be 1, 2, 3, or 4.

buffer – A pointer to the start of the physically contiguous data buffer that will be transmitted. The buffer should be at least *count* bytes long.

count – The number of bytes to transfer.

id – A pointer to the location that will hold the unique transfer identifier that is assigned to this transfer. This value can be used to determine when the transfer has completed.

GscSio4ChannelQueryTransfer

GscSio4ChannelQueryTransfer(…) is used to determine the status of a transfer that was initiated by a call to either GscSio4ChannelReceiveData (…) or GscSio4ChannelTransmitData (…). The result is returned in *stat* and will be 0 if the transfer has completed or non-zero if it has not completed.

Supported Hardware:

All

Prototype:

```
int GscSio4ChannelQueryTransfer(
                                    int boardNumber,
                                    int channel,
                                    int *stat,
                                    int id);
```

Parameters:

boardNumber – The number of the desired board. This number corresponds to the results of the GscFindBoards(…) function. Note that this number will always be 1 in a single board system.

channel – The desired channel number. This number will be 1, 2, 3, or 4.

stat – A pointer to the location that will hold the returned status of the transfer. The *stat* will be 0 if the transfer has completed. Otherwise, it will hold the number of bytes left to transfer.

id – The unique ID that was assigned to the transfer by the call to either GscSio4ChannelReceiveData(…) or GscSio4ChannelTransmitData(…)

GscSio4ChannelWaitForTransfer

GscSio4ChannelWaitForTransfer (…) is used to wait for the completion of a transfer that was initiated by a call to either GscSio4ChannelReceiveData (…) or GscSio4ChannelTransmitData (…). The routine will return when either the transfer completes or the timeout period expires. If the timeout period expires, the *bytesTransferred* parameter will be updated with the number of bytes that were successfully transferred. If the transfer completes, or another type of error occurs, the *bytesTransferred* parameter will be -1.

Supported Hardware:

All

Prototype:

```
int GscSio4ChannelWaitForTransfer(
                                    int boardNumber,
                                    int channel,
                                    int timeout,
                                    int id,
                                    int *bytesTransferred);
```

Parameters:

boardNumber – The number of the desired board. This number corresponds to the results of the GscFindBoards(…) function. Note that this number will always be 1 in a single board system.

channel – The desired channel number. This number will be 1, 2, 3, or 4.

timeout – The desired amount of time in milliseconds that the routine will wait for the transfer to complete.

id – The unique ID that was assigned to the transfer by the call to either GscSio4ChannelReceiveData(…) or GscSio4ChannelTransmitData(…)

bytesTransferred - A pointer to the location that will hold the number of bytes that were actually transferred if the timeout period expires. This value will be -1 if the transfer completes or an error occurs.

GscSio4ChannelFlushTransfer

GscSio4ChannelFlushTransfer (…) is used to force any data that is in the Rx FIFO to be transferred via DMA to memory. For a Tx channel, data is transferred to the Tx FIFO until it is full. Calling this routine is only necessary when a transfer did not complete on its own, or when aborting a transfer that has not completed.

Supported Hardware:

All

Prototype:

```
int GscSio4ChannelFlushTransfer(
                                    int boardNumber,
                                    int channel,
                                    int id);
```

Parameters:

boardNumber – The number of the desired board. This number corresponds to the results of the GscFindBoards(…) function. Note that this number will always be 1 in a single board system.

channel – The desired channel number. This number will be 1, 2, 3, or 4.

id – The unique ID that was assigned to the transfer by the call to either GscSio4ChannelReceiveData(…) or GscSio4ChannelTransmitData(…)

GscSio4ChannelRemoveTransfer

GscSio4ChannelRemoveTransfer (…) is used to remove a pending transfer from the transfer queue. Calling this routine is only necessary when a transfer did not complete on its own, or when aborting a transfer that has not completed. If a transfer ID of -1 is passed to the routine, all pending transfers will be removed.

Supported Hardware:

All

Prototype:

```
int GscSio4ChannelRemoveTransfer(
                                    int boardNumber,
                                    int channel,
                                    int id,
                                    int *bytesTransferred);
```

Parameters:

boardNumber – The number of the desired board. This number corresponds to the results of the GscFindBoards(…) function. Note that this number will always be 1 in a single board system.

channel – The desired channel number. This number will be 1, 2, 3, or 4.

id – The unique ID that was assigned to the transfer by the call to either GscSio4ChannelReceiveData(…) or GscSio4ChannelTransmitData(…)

bytesTransferred - A pointer to the location that will hold the number of bytes that were actually transferred before the call to GscSio4ChannelRemoveTransfer (). This value will be -1 if the transfer had already completed or an error occurs.

GscFindBoardsGscSio4ChannelRegisterInterrupt

GscSio4ChannelRegisterInterrupt (…) is used register a callback routine with the interrupt handler. There are several interrupt sources associated with each interrupt. This routine allows any or all of the interrupt sources to be associated with a callback function. The callback function can be shared between interrupt sources or a different callback can be used for each source. This routine also determines whether the interrupt occurs on the Rising Edge (High True) or Falling Edge (Low True).

Supported Hardware:

All

Prototype:

```
int GscSio4ChannelRegisterInterrupt(
                                        int boardNumber,
                                        int channel,
                                        int interrupt,
                                        int type,
                                        GSC_CB_FUNCTION *function);
```

Parameters:

boardNumber – The number of the desired board. This number corresponds to the results of the GscFindBoards(…) function. Note that this number will always be 1 in a single board system.

channel – The desired channel number. This number will be 1, 2, 3, or 4.

interrupt – This value determines which interrupts are associated with the provided callback function. This value should be the logical OR of one or more of the following:

GSC_INTR_SYNC_DETECT – Triggers an interrupt when the SYNC byte is received on the channel. (This source is not available on the –Sync boards)

GSC_INTR_USC – Triggers an interrupt when the on board USC has an interrupt pending. Refer to the USC data sheet for details of its possible interrupt sources. (This source is not available on the –Sync boards)

GSC_INTR_TX_FIFO_EMPTY – Triggers an interrupt when the Transmit FIFO for the channel is empty.

GSC_INTR_TX_FIFO_FULL – Triggers an interrupt when the Transmit FIFO for the channel is full.

GSC_INTR_TX_FIFO_ALMOST_EMPTY – Triggers an interrupt when the Transmit FIFO for the channel is almost empty. The level at which this

interrupt will occur is set by calling the GscSio4ChannelSetTxAlmost(…) routine.

GSC_INTR_RX_FIFO_EMPTY – Triggers an interrupt when the Receive FIFO for the channel is empty.

GSC_INTR_RX_FIFO_FULL – Triggers an interrupt when the Receive FIFO for the channel is full.

GSC_INTR_RX_FIFO_ALMOST_FULL – Triggers an interrupt when the Receive FIFO for the channel is almost full. The level at which this interrupt will occur is set by calling the GscSio4ChannelSetRxAlmost(…) routine.

GSC_INTR_RX_ENVELOPE – Triggers an interrupt when the RX Envelope signal changes. (This source is only available on the –Sync boards)

type – This value determines whether the interrupt occurs on the rising of falling edge. It should be one of the following:

GSC_RISING_EDGE – The interrupt will occur on the rising edge of the interrupt signal (i.e. when the condition becomes true.)

GSC_FALLING_EDGE – The interrupt will occur on the falling edge of the interrupt signal (i.e. when the condition becomes not true.)

function – This is the address of the interrupt callback function. If this value is set to NULL, the callback for the current "interrupt" parameter will be cleared, otherwise this routine will be called for each of the conditions specified in the "interrupt" parameter. The prototype for the callback function is:

```
void CALLBACK callback_function(
                                    int boardNumber,
                                    int channel,
                                    int interrupt);
```

The parameters to the callback specify the board and channel number on which the interrupt occurred as well as the source of the interrupt (as defined above.) If multiple interrupt sources are mapped to the same callback routine, the "interrupt" parameter can be used to determine the source of the interrupt.

GscSio4ChannelSetClock

GscSio4ChannelSetClock(…) is used to set the serial bit rate (baud rate) for a specific channel. Under normal conditions, this routine will not be used since the GscSio4ChannelSetMode(…) function sets the bit rate of the channel when the channel's mode is set. This function is provided to allow the bit rate to be changed without re-configuring the channel.

Supported Hardware:

All

Prototype:

```
int GscSio4ChannelSetClock(
                                    int boardNumber,
                                    int channel,
                                    int frequency);
```

Parameters:

boardNumber – The number of the desired board. This number corresponds to the results of the GscFindBoards(…) function. Note that this number will always be 1 in a single board system.

channel – The desired channel number. This number will be 1, 2, 3, or 4.

frequency – The desired bit rate for this channel. This value is specified in Hz and can range from 100 to 10000000 (1000000 for async channels).

Protocol Level Routines

The Protocol Level Routines perform functions that apply to a specific protocol on a single channel on an SIO4 board. Each of these routines requires the board number (boardNumber) as the first parameter and the channel number (channel) as the second parameter. The board number corresponds to the results of the GscFindBoards(…) function. Note that this number will always be 1 in a single board system. The channel number will always be 1, 2, 3, or 4.

These routines can be called at any time. All of these functions return zero if successful or a non-zero error code if a failure occurs.

GscSio4HdlcGetDefaults

GscSio4HdlcGetDefaults(…) returns the default HDLC configuration structure.

Supported Hardware:

PCI-SIO4B

Prototype:

```
int GscSio4HdlcGetDefaults(
                                    PGSC_HDLC_CONFIG config);
```

Parameters:

config – A pointer to a configuration structure that will be filled in with default configuration values.

GscSio4HdlcSetConfig / GscSio4HdlcGetConfig

GscSio4HdlcSetConfig(…) sets the mode of the specified channel to HDLC and sets the current configuration to the values specified in the *config* parameter.

Supported Hardware:

PCI-SIO4B

Prototype:

```
int GscSio4HdlcSetConfig(
                                int boardNumber,
                                int channel,
                                GSC_HDLC_CONFIG config);

int GscSio4HdlcGetConfig(
                                int boardNumber,
                                int channel,
                                PGSC_HDLC_CONFIG config);
```

Parameters:

boardNumber – The number of the desired board. This number corresponds to the results of the GscFindBoards(…) function. Note that this number will always be 1 in a single board system.

channel – The desired channel number. This number will be 1, 2, 3, or 4.

config – The desired/current configuration structure for the channel.

GscSio4AsyncGetDefaults

GscSio4AsyncGetDefaults(…) returns the default Async configuration structure.

Supported Hardware:

PCI-SIO4B

Prototype:

```
int GscSio4AsyncGetDefaults(
                                    PGSC_ASYNC_CONFIG config);
```

Parameters:

config – A pointer to a configuration structure that will be filled in with default configuration values.

GscSio4AsyncSetConfig / GscSio4AsyncGetConfig

GscSio4AsyncSetConfig(…) sets the mode of the specified channel to Async and sets the current configuration to the values specified in the *config* parameter.

Supported Hardware:

PCI-SIO4B

Prototype:

```
int GscSio4AsyncSetConfig(
                                int boardNumber,
                                int channel,
                                GSC_ASYNC_CONFIG config);

int GscSio4AsyncGetConfig(
                                int boardNumber,
                                int channel,
                                PGSC_ASYNC_CONFIG config);
```

Parameters:

boardNumber – The number of the desired board. This number corresponds to the results of the GscFindBoards(…) function. Note that this number will always be 1 in a single board system.

channel – The desired channel number. This number will be 1, 2, 3, or 4.

config – The desired/current configuration structure for the channel.

GscSio4BiSyncGetDefaults

GscSio4BiSyncGetDefaults(…) returns the default BiSync configuration structure.

Supported Hardware:

PCI-SIO4B

Prototype:

```
int GscSio4BiSyncGetDefaults(
                                    PGSC_BISYNC_CONFIG config);
```

Parameters:

config – A pointer to a configuration structure that will be filled in with default configuration values.

GscSio4BiSyncSetConfig / GscSio4BiSyncGetConfig

GscSio4BiSyncSetConfig(…) sets the mode of the specified channel to bisync and sets the current configuration to the values specified in the *config* parameter.

Supported Hardware:

PCI-SIO4B

Prototype:

```
int GscSio4BiSyncSetConfig(
                                    int boardNumber,
                                    int channel,
                                    GSC_BISYNC_CONFIG config);

int GscSio4BiSyncGetConfig(
                                    int boardNumber,
                                    int channel,
                                    PGSC_BISYNC_CONFIG config);
```

Parameters:

boardNumber – The number of the desired board. This number corresponds to the results of the GscFindBoards(…) function. Note that this number will always be 1 in a single board system.

channel – The desired channel number. This number will be 1, 2, 3, or 4.

config – The desired/current configuration structure for the channel.

GscSio4SyncGetDefaults

GscSio4SyncGetDefaults(…) returns the default Sync configuration structure.

Supported Hardware:

PCI-SIO4B-SYNC

Prototype:

```
int GscSio4SyncGetDefaults(
                                    PGSC_SYNC_CONFIG config);
```

Parameters:

config – A pointer to a configuration structure that will be filled in with default configuration values.

GscSio4SyncSetConfig / GscSio4SyncGetConfig

GscSio4SyncSetConfig(…) sets the mode of the specified channel to Sync and sets the current configuration to the values specified in the *config* parameter.

Supported Hardware:

PCI-SIO4B-SYNC

Prototype:

```
int GscSio4SyncSetConfig(
                                int boardNumber,
                                int channel,
                                GSC_SYNC_CONFIG config);

int GscSio4SyncGetConfig(
                                int boardNumber,
                                int channel,
                                PGSC_SYNC_CONFIG config);
```

Parameters:

boardNumber – The number of the desired board. This number corresponds to the results of the GscFindBoards(…) function. Note that this number will always be 1 in a single board system.

channel – The desired channel number. This number will be 1, 2, 3, or 4.

config – The desired/current configuration structure for the channel.

GscSio4BiSync16GetDefaults

GscSio4BiSync16GetDefaults(…) returns the default bisync16 configuration structure.

Supported Hardware:

PCI-SIO4B-BISYNC

Prototype:

```
int GscSio4BiSync16GetDefaults(
                                    PGSC_BISYNC16_CONFIG config);
```

Parameters:

config – A pointer to a configuration structure that will be filled in with default configuration values.

GscSio4BiSync16SetConfig / GscSio4BiSync16GetConfig

GscSio4BiSync16SetConfig(…) sets the mode of the specified channel to bisync16 and sets the current configuration to the values specified in the *config* parameter.

Supported Hardware:

PCI-SIO4B-BISYNC

Prototype:

```
int GscSio4BiSync16SetConfig(
                                int boardNumber,
                                int channel,
                                GSC_BISYNC16_CONFIG config);

int GscSio4BiSync16GetConfig(
                                int boardNumber,
                                int channel,
                                PGSC_BISYNC16_CONFIG config);
```

Parameters:

boardNumber – The number of the desired board. This number corresponds to the results of the GscFindBoards(…) function. Note that this number will always be 1 in a single board system.

channel – The desired channel number. This number will be 1, 2, 3, or 4.

config – The desired/current configuration structure for the channel.

GscSio4BiSync16GetTxCounts

GscSio4BiSync16GetTxCounts(…) is used to retrieve the initial and remaining Tx counts for a channel configured in bisync16 mode.

Supported Hardware:

PCI-SIO4B-BISYNC

Prototype:

```
int GscSio4BiSync16GetTxCounts(
                                    int boardNumber,
                                    int channel,
                                    int *remaining,
                                    int *initial);
```

Parameters:

boardNumber – The number of the desired board. This number corresponds to the results of the GscFindBoards(…) function. Note that this number will always be 1 in a single board system.

channel – The desired channel number. This number will be 1, 2, 3, or 4.

remaining – The remaining Tx counts value.

initial – The initial Tx Counts value.

GscSio4BiSync16GetRxCounts

GscSio4BiSync16GetRxCounts(…) is used to retrieve the initial and remaining Rx counts for a channel configured in bisync16 mode.

Supported Hardware:

PCI-SIO4B-BISYNC

Prototype:

```
int GscSio4BiSync16GetRxCounts(
                                    int boardNumber,
                                    int channel,
                                    int *remaining,
                                    int *initial);
```

Parameters:

boardNumber – The number of the desired board. This number corresponds to the results of the GscFindBoards(…) function. Note that this number will always be 1 in a single board system.

channel – The desired channel number. This number will be 1, 2, 3, or 4.

remaining – The remaining Rx counts value.

initial – The initial Rx Counts value.

GscSio4BiSync16EnterHuntMode

GscSio4BiSync16EnterHuntMode(…) is used to cause a channel configured in bisync16 mode to enter hunt mode.

Supported Hardware:

PCI-SIO4B-BISYNC16??

Prototype:

```
int GscSio4BiSync16EnterHuntMode(
                                    int boardNumber,
                                    int channel)
```

Parameters:

boardNumber – The number of the desired board. This number corresponds to the results of the GscFindBoards(…) function. Note that this number will always be 1 in a single board system.

channel – The desired channel number. This number will be 1, 2, 3, or 4.

GscSio4BiSync16AbortTx

GscSio4BiSync16AbortTx(…) is used to cause a channel configured in bisync16 mode to abort the current transmission.

Supported Hardware:

PCI-SIO4B-BISYNC16??

Prototype:

int GscSio4BiSync16AbortTx(
int boardNumber,
int channel)

Parameters:

boardNumber – The number of the desired board. This number corresponds to the results of the GscFindBoards(…) function. Note that this number will always be 1 in a single board system.

channel – The desired channel number. This number will be 1, 2, 3, or 4.

GscSio4BiSync16Pause

GscSio4BiSync16Pause(…) is used to cause a channel configured in bisync16 mode to pause the current transmission.

Supported Hardware:

PCI-SIO4B-BISYNC16??

Prototype:

int GscSio4BiSync16Pause(
int boardNumber,
int channel)

Parameters:

boardNumber – The number of the desired board. This number corresponds to the results of the GscFindBoards(…) function. Note that this number will always be 1 in a single board system.

channel – The desired channel number. This number will be 1, 2, 3, or 4.

GscSio4BiSync16Resume

GscSio4BiSync16Resume(…) is used to cause a channel configured in bisync16 mode to pause the current transmission.

Supported Hardware:

PCI-SIO4B-BISYNC16??

Prototype:

```
int GscSio4BiSync16Resume(
                                    int boardNumber,
                                    int channel)
```

Parameters:

boardNumber – The number of the desired board. This number corresponds to the results of the GscFindBoards(…) function. Note that this number will always be 1 in a single board system.

channel – The desired channel number. This number will be 1, 2, 3, or 4.

Structures and Macro Definitions

This section contains the descriptions of the various structures and macro definitions available to users of the API.

Devices Structure

```
typedef struct
{
        int     busNumber;          // Identifies the bus that contains the board
        int     slotNumber;         // Identifies the slot that contains the board
        int     vendorId;           // Identifies the board Vendor
        int     deviceId;           // Identifies the device
        char    serialNumber[25];   // A unique board serial number
} GSC_DEVICES_STRUCT;
```

Interrupt Callback Prototype

```
typedef void
((CALLBACK *GSC_CB_FUNCTION)(int boardNumber, int channel, int interrupt));
```

For the Linux platform, the macro CALLBACK is null. On the Win32 platform, this macro declares the function calling convention as __stdcall, which is required by Microsoft .Net 2003 applications.

Channel Mode Definitions

The Channel Mode Definitions are used to set the current operating protocol for each channel of the SIO4 board. These definitions are passed as a parameter of the GscSio4ChannelSetMode(…) command.

Macro	Protocol	Defaults
GSC_MODE_ASYNC	Asynchronous Mode	8 Data Bits No Parity 1 Stop Bit 16x Clock NRZ Encoding
GSC_MODE_HDLC	HDLC/SDLC Mode	8 Data Bits NRZ Encoding
GSC_MODE_SYNC	Synchronous Mode*	8 Data Bits 0 Gap Bits NRZ Encoding
GSC_MODE_SYNC_ENV	Synchronous Mode w/ Envelope*	8 Data Bits 0 Gap Bits NRZ Encoding
GSC_MODE_ISO	Isochronous Mode	8 Data Bits NRZ Encoding
GSC_MODE_MONOSYNC	Monosync Mode	8 Data Bits NRZ Encoding
GSC_MODE_BISYNC	BiSync Mode	8 Data Bits NRZ Encoding
GSC_MODE_TRANS_BISYNC	Transparent BiSync Mode	8 Data Bits NRZ Encoding
GSC_MODE_802_3	IEEE 802.3 Ethernet Mode	8 Data Bits NRZ Encoding

* These are the only modes that are available on the –SYNC card. They are not available on the standard card.

Channel Mode Configuration Structures

The Channel Mode Configuration structures are used by the GscApi mode configuration functions that correspond with each mode. For example, the structure GSC_HDLC_CONFIG is used by the GscApi configuration functions as follows:

```
GSC_HDLC_CONFIG cfg;                          // declare a configuration variable

GscSio4HdlcGetDefaults(&cfg);                 // get the default settings for Hdlc
mode

GscSio4HdlcSetConfig(board, channel, cfg);    // configure a channel in Hdlc mode

GscSio4HdlcGetConfig(board, channel, &cfg);   // retrieve current configuration
```

The GSC_HDLC_CONFIG structure definition, along with the structures corresponding to the Async, BiSync, Sync and BiSync16 modes are listed below.

GSC_ASYNC_CONFIG Structure

```
typedef struct _GSC_ASYNC_CONFIG
{
        // Channel Configuration Variables
        int     bitRate;                      // Baud rate for the channel.
        int encoding;                         // Encoding - NRZ, BiPhase, etc.
        int protocol;                         // Bus Protocol - RS485, RS232, V.35, etc.
        int termination;                      // Termination Resistors enabled/disabled
        int parity;                           // Parity mode - None, Even, Odd, etc.
        int stopBits;                         // Stop bits - 0, 1, 1.5, 2

        // Transmitter Configuration Variables
        int txStatus;                         // Transmitter Enabled/Disabled
        int txCharacterLength;                // Bits per Tx character
        int txClockSource;                    // Clock source for the transmitter

        // Receiver Configuration Variables
        int rxStatus;                         // Receiver Enabled/Disabled
        int rxCharacterLength;                // Bits per Rx character
        int rxClockSource;                    // Clock source for the receiver

        // Pin Configuration Variables
        int interfaceMode;                    // DTE or DCE interface
        int txDataPinMode;                    // Auto (system use) or GPIO
        int rxDataPinMode;                    // Auto (system use) or GPIO
        int txClockPinMode;                   // Auto (system use) or GPIO
        int rxClockPinMode;                   // Auto (system use) or GPIO
        int ctsPinMode;                       // Auto (system use) or GPIO
        int rtsPinMode;                       // Auto (system use) or GPIO
        int loopbackMode;                     // None, internal, or external loop back

} GSC_ASYNC_CONFIG, *PGSC_ASYNC_CONFIG;
```

GSC_HDLC_CONFIG Structure

```
typedef struct _GSC_HDLC_CONFIG
{
        // Channel Configuration Variables
        int bitRate;                            // Baud rate for the channel.
        int encoding;                           // Encoding - NRZ, BiPhase, etc.
        int protocol;                           // Bus Protocol - RS485, RS232, V.35, etc.
        int termination;                        // Termination Resistors enabled/disabled
        int parity;                             // Parity mode - None, Even, Odd, etc.
        int crcMode;                            // CRC Type - Disabled, CCITT, etc.
        int crcInitialValue;                    // Initial CRC - All 1 or 0

        // Transmitter Configuration Variables
        int txStatus;                           // Transmitter Enabled/Disabled
        int txCharacterLength;                  // Bits per Tx character

        int txUnderRun;                         // What to do on a Tx underrun
        int txPreamble;                         // Length of Preamble
        int txPreamblePattern;                  // Type of Preamble
        int txSharedZero;                       // Share 0s in adjacent flags?
        int txClockSource;                      // Clock source for the transmitter
        int txIdleCondition;                    // What to transmit when the line is idle

        // Receiver Configuration Variables
        int rxStatus;                           // Receiver Enabled/Disabled
        int rxCharacterLength;                  // Bits per Rx character
        int rxAddrSearchMode;                   // Rx address search mode
        int rxAddress;                          // Address to search for
        int rxClockSource;                      // Clock source for the receiver
        int rxReceiveStatusBlocks;              // Enable/disable 16 bit receive
                                                // status blocks in RX FIFO

        // Pin Configuration Variables
        int interfaceMode;                      // DTE or DCE interface
        int txDataPinMode;                      // Auto (system use) or GPIO
        int rxDataPinMode;                      // Auto (system use) or GPIO
        int txClockPinMode;                     // Auto (system use) or GPIO
        int rxClockPinMode;                     // Auto (system use) or GPIO
        int ctsPinMode;                         // Auto (system use) or GPIO
        int rtsPinMode;                         // Auto (system use) or GPIO
        int loopbackMode;                       // None, internal, or external loop back

        // Misc Configuration Variables
        int packetFraming;                      // Internal use only, leave enabled

} GSC_HDLC_CONFIG, *PGSC_HDLC_CONFIG;
```

GSC_BISYNC_CONFIG Structure

```
typedef struct _GSC_BISYNC_CONFIG
{
        // Channel Configuration Variables
        int bitRate;                            // Baud rate for the channel.
        int encoding;                           // Encoding - NRZ, BiPhase, etc.
        int protocol;                           // Bus Protocol - RS485, RS232, V.35, etc.
        int termination;                        // Termination Resistors enabled/disabled
        int parity;                             // Parity mode - None, Even, Odd, etc.
        int crcMode;                            // CRC Type - Disabled, CCITT, etc.
        int crcInitialValue;                    // Initial CRC - All 1 or 0

        // Transmitter Configuration Variables
        int txStatus;                           // Transmitter Enabled/Disabled
        int txCharacterLength;                  // Bits per Tx character
        int txClockSource;                      // Clock source for the transmitter
        int txIdleCondition;                    // What to transmit when the line is idle
        int txSyncWord;                         // Two character sync pattern
        int txUnderRun;                         // What to do on a Tx underrun
        int txPreamble;                         // Enable/disable preamble before sync open
        int txPreambleLength;                   // Preamble length - 8,16,32,64 bits
        int txPreamblePattern;                  // Preamble pattern - all zeros, all
                                                // ones, etc.
        int txShortSync;                        // Length of sync character -
                                                // 8 bits or same as txCharacterLength

        // Receiver Configuration Variables
        int rxStatus;                           // Receiver Enabled/Disabled
        int rxClockSource;                      // Clock source for the receiver
        int rxCharacterLength;                  // Bits per Rx character
        int rxSyncWord;                         // Two character sync pattern
        int rxSyncStrip;                        // Sync character stripping enable/disable
        int rxShortSync;                        // Length of sync character - 8 bits or same
                                                // as rxCharacterLength

        // Pin Configuration Variables
        int interfaceMode;                      // DTE or DCE interface
        int txDataPinMode;                      // Auto (system use) or GPIO
        int rxDataPinMode;                      // Auto (system use) or GPIO
        int txClockPinMode;                     // Auto (system use) or GPIO
        int rxClockPinMode;                     // Auto (system use) or GPIO
        int ctsPinMode;                         // Auto (system use) or GPIO
        int rtsPinMode;                         // Auto (system use) or GPIO
        int loopbackMode;                       // None, internal, or external loop back

        // Misc Configuration Variables
        int packetFraming;                      // Internal use only, leave disabled
} GSC_BISYNC_CONFIG, *PGSC_BISYNC_CONFIG;
```

GSC_SYNC_CONFIG Structure

```
typedef struct _GSC_SYNC_CONFIG
{
        // Channel Configuration Variables
        int bitRate;                        // Baud rate for the channel.
        int encoding;                       // Encoding - NRZ, NRZB
        int protocol;                       // Bus Protocol - RS485, RS232, V.35, etc.
        int termination;                    // Termination Resistors enabled/disabled

        // Transmitter Configuration Variables
        int txStatus;                       // Transmitter Enabled/Disabled
        int txCharacterLength;              // Bits per Tx character
        int txGapLength;                    // Bits between Tx characters
        int txClockSource;                  // Clock source for the transmitter
        int txClockEdge;                    // Clock edge for the transmitter
        int txEnvPolarity;                  // Envelope polarity for the transmitter
        int txIdleCondition;                // What to transmit when the line is idle
        int txClockIdleCondition;           // What to do with the clock when line idle
        int txMsbLsb;                       // Bit order for transmitter

        // Receiver Configuration Variables
        int rxStatus;                       // Receiver Enabled/Disabled
        int rxClockSource;                  // Clock source for the receiver
        int rxClockEdge;                    // Clock edge for the receiver
        int rxEnvPolarity;                  // Envelope polarity for the receiver
        int rxMsbLsb;                       // Bit order for receiver

        // Pin Configuration Variables
        int interfaceMode;                  // DTE or DCE interface
        int txDataPinMode;                  // Auto (system use) or GPIO
        int rxDataPinMode;                  // Auto (system use) or GPIO
        int txClockPinMode;                 // Auto (system use) or GPIO
        int rxClockPinMode;                 // Auto (system use) or GPIO
        int txEnvPinMode;                   // Auto (system use) or GPIO
        int rxEnvPinMode;                   // Auto (system use) or GPIO
        int loopbackMode;                   // None, internal, or external loop back

        // Misc Configuration Variables
        int packetFraming;                  // Internal use only, leave disabled

} GSC_SYNC_CONFIG, *PGSC_SYNC_CONFIG;
```

GSC_BISYNC16_CONFIG Structure

```
typedef struct _GSC_BISYNC16_CONFIG
{
        // Channel Configuration Variables
        int bitRate;                        // Baud rate for the channel.
        int encoding;                       // Encoding - NRZ, BiPhase, etc.
        int protocol;                       // Bus Protocol - RS485, RS232, V.35, etc.
        int termination;                    // Termination Resistors enabled/disabled

        // Transmitter Configuration Variables
        int txStatus;                       // Transmitter Enabled/Disabled
        int txIdleCondition;                // What to transmit when the line is idle
        int txSyncWord;                     // Two character sync pattern
        int txBitOrder;
        int txByteOrder;

        // Receiver Configuration Variables
        int rxStatus;                       // Receiver Enabled/Disabled
        int rxSyncWord;                     // Two character sync pattern
        int maxRxCount;

        // Pin Configuration Variables
        int interfaceMode;                  // DTE or DCE interface
        int txDataPinMode;                  // Auto (system use) or GPIO
        int rxDataPinMode;                  // Auto (system use) or GPIO
        int txClockPinMode;                 // Auto (system use) or GPIO
        int rxClockPinMode;                 // Auto (system use) or GPIO
        int ctsPinMode;                     // Auto (system use) or GPIO
        int rtsPinMode;                     // Auto (system use) or GPIO
        int loopbackMode;                   // None, internal, or external loop back

} GSC_BISYNC16_CONFIG, *PGSC_BISYNC16_CONFIG;
```

Channel Encoding Definitions

The Channel Encoding Definitions are used to set the desired channel encoding for each channel of the SIO4 board. These definitions are passed as a parameter of the GscSio4ChannelSetEncoding(…) command.

Macro	Description
GSC_ENCODING_NRZ	
GSC_ENCODING_NRZB	
GSC_ENCODING_NRZI_MARK	
GSC_ENCODING_NRZI_SPACE	
GSC_ENCODING_BIPHASE_MARK	
GSC_ENCODING_BIPHASE_SPACE	
GSC_ENCODING_BIPHASE_LEVEL	
GSC_ENCODING_DIFF_BIPHASE_LEVEL	

Channel Protocol and Termination Definitions

GSC_PROTOCOL_RS422_RS485,
GSC_PROTOCOL_RS423,
GSC_PROTOCOL_RS232,
GSC_PROTOCOL_RS530_1,
GSC_PROTOCOL_RS530_2,
GSC_PROTOCOL_V35_1,
GSC_PROTOCOL_V35_2,
GSC_PROTOCOL_RS422_RS423_1,
GSC_PROTOCOL_RS422_RS423_2,

GSC_TERMINATION_ENABLED,
GSC_TERMINATION_DISABLED,

Channel Interrupt Definitions

```
GSC_INTR_RISING_EDGE,
GSC_INTR_FALLING_EDGE,
GSC_INTR_HIGH_TRUE,
GSC_INTR_LOW_TRUE,

GSC_INTR_SYNC_DETECT                = 0x0001,
GSC_INTR_USC                        = 0x0002,
GSC_INTR_TX_FIFO_EMPTY              = 0x0004,
GSC_INTR_TX_FIFO_FULL               = 0x0008,
GSC_INTR_TX_FIFO_ALMOST_EMPTY       = 0x0010,
GSC_INTR_RX_FIFO_EMPTY              = 0x0020,
GSC_INTR_RX_FIFO_FULL               = 0x0040,
GSC_INTR_RX_FIFO_ALMOST_FULL        = 0x0080,
GSC_INTR_TX_TRANSFER_COMPLETE       = 0x0100,
GSC_INTR_RX_TRANSFER_COMPLETE       = 0x0200,
GSC_INTR_RX_ENVELOPE                = GSC_INTR_SYNC_DETECT,
// -Sync card definition
```

Channel Pin Definitions

```
GSC_PIN_DTE,
GSC_PIN_DCE,
GSC_PIN_AUTO,
GSC_PIN_GPIO,
GSC_PIN_RX_CLOCK,       // Keep these enums in order
GSC_PIN_RX_DATA,        //
GSC_PIN_CTS,            //
GSC_PIN_DCD,            //
GSC_PIN_TX_CLOCK,       //
GSC_PIN_TX_DATA,        //
GSC_PIN_RTS,            //
GSC_PIN_AUXCLK,         // Keep these enums in order
GSC_PIN_RX_ENV,
GSC_PIN_TX_ENV,
```

Channel Parity Definitions

GSC_PARITY_NONE,
GSC_PARITY_EVEN,
GSC_PARITY_ODD,
GSC_PARITY_MARK,
GSC_PARITY_SPACE,

Channel Stop Bits Definition

GSC_STOP_BITS_0,
GSC_STOP_BITS_1,
GSC_STOP_BITS_1_5,
GSC_STOP_BITS_2,

Loopback Definitions

GSC_LOOP_NONE,
GSC_LOOP_INTERNAL,
GSC_LOOP_EXTERNAL,

HDLC CRC Definitions

GSC_CRC_NONE,
GSC_CRC_16,
GSC_CRC_32,
GSC_CRC_CCITT,
GSC_CRC_INIT_0,
GSC_CRC_INIT_1,

Local Register Definitions

The Local Register Definitions are used to access the various registers that are contained in the on board FPGA. These registers should not be accessed during normal operation and are included only for diagnostic purposes. For detailed descriptions of the registers, refer to the SIO4 hardware user's manual.

Macro	Value	Description
FW_REVISION_REG	0x0000	Firmware Revision Register
BOARD_CONTROL_REG	0x0004	Board Control Register
BOARD_STATUS_REG	0x0008	Board Status Register
CLOCK_CONTROL_REG	0x000c	Clock Control Register
TX_ALMOST_BASE_REG	0x0010	Base value for the Tx Almost registers
RX_ALMOST_BASE_REG	0x0014	Base value for the Rx Almost registers
DATA_FIFO_BASE_REG	0x0018	Base value for the Tx and Rx Data FIFOs
CONTROL_STATUS_BASE_REG	0x001c	Base value for the Control/Status registers
SYNC_CHARACTER_BASE_REG	0x0050	Base value for the Sync Byte Registers
INTERRUPT_CONTROL_REG	0x0060	Interrupt Control Register
INTERRUPT_STATUS_REG	0x0064	Interrupt Status/Clear Register
INTERRUPT_EDGE_LEVEL_REG	0x0068	Interrupt Edge/Level Register (RO)
INTERRUPT_HI_LO_REG	0x006c	Interrupt High/Low, Rising/Falling register
PIN_SOURCE_BASE_REG	0x0080	Base value for the Pin Source Registers
PIN_STATUS_BASE_REG	0x0090	Base value for the Pin Status Registers
POSC_RAM_ADDRESS_REG	0x00a0	Programmable OSC Address Register
POSC_RAM_DATA_REG	0x00a4	Programmable OSC Data Register
POSC_CONTROL_STATUS_REG	0x00a8	Programmable OSC Control/Status Register
TX_COUNT_BASE_REG	0x00b0	
FIFO_COUNT_BASE_REG	0x00d0	Base value for the FIFO Count Registers
FIFO_SIZE_BASE_REG	0x00e0	Base value for the FIFO Size Registers
FEATURES_REG	0x00fc	Features Register

Channel Register Definitions

The Channel Register Definitions are used to access the various registers that are contained in the Zilog USC chip for each channel. These registers should not be accessed during normal operation and are included only for diagnostic purposes. For detailed descriptions of the registers, refer to the Zilog USC hardware user's manual.

Macro	Value	Description
USC_CCAR	0x0000	Channel Command/Address Register
USC_CMR	0x0002	Channel Mode Register
USC_CCSR	0x0004	Channel Command/Status Register
USC_CCR	0x0006	Channel Control Register
USC_TMDR	0x000c	Test Mode Data Register
USC_TMCR	0x000e	Test Mode Control Register
USC_CMCR	0x0010	Clock Mode Control Register
USC_HCR	0x0012	Hardware Configuration Register
USC_IVR	0x0014	Interrupt Vector Register
USC_IOCR	0x0016	I/O Control Register
USC_ICR	0x0018	Interrupt Control Register
USC_DCCR	0x001a	Daisy Chain Control Register
USC_MISR	0x001c	Misc. Interrupt Status Register
USC_SICR	0x001e	Status Interrupt Control Register
USC_RDR	0x0020	Receive Data Register (RO)
USC_RMR	0x0022	Receive Mode Register
USC_RCSR	0x0024	Receive Command Status Register
USC_RICR	0x0026	Receive Interrupt Control Register
USC_RSR	0x0028	Receive Sync Register
USC_RCLR	0x002a	Receive Count Limit Register
USC_RCCR	0x002c	Receive Character Count Register
USC_TC0R	0x002e	Time Constant 0 Register
USC_TDR	0x0030	Transmit Data Register (WO)
USC_TMR	0x0032	Transmit Mode Register
USC_TCSR	0x0034	Transmit Command Status Register
USC_TICR	0x0036	Transmit Interrupt Control Register
USC_TSR	0x0038	Transmit Sync Register
USC_TCLR	0x003a	Transmit Count Limit Register
USC_TCCR	0x003c	Transmit Character Count Register
USC_TC1R	0x003e	Time Constant 1 Register

Miscellaneous Token Definitions

GSC_ENABLED,
GSC_DISABLED,

GSC_CLOCK_INTERNAL,
GSC_CLOCK_EXTERNAL,

GSC_LSB_FIRST,
GSC_MSB_FIRST,