SIO4/8

Four/Eight Channel High Speed Serial I/O

All SIO4 and SIO8 Models
All Form Factors
All Standard Fast Async Firmware Versions

Fast Async Protocol Library Reference Manual

Manual Revision: March 19, 2024

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

Preface

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

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

Table of Contents

Table of Figures

1. Introduction

This document provides information on the Fast Async Protocol Library, which is a library designed to facilitate use of the Fast Async versions of the SIO4 and SIO8.

1.1. Purpose

The purpose of this document is twofold. First, it is intended to give a basic description of the Fast Async protocol implementation of the SIO4. Second, it is intended to give a complete description of the Fast Async Protocol Library programming interface.

1.2. Acronyms

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

Acronyms	Description
CTS	Clear To Send (This is a cable interface signal used for hardware flow control.)
DCE	Data Communications Equipment (This is a configuration option that affects the organization of signals at the cable interface.)
DMA	Direct Memory Access
DTE	Data Terminal Equipment (This is a configuration option that affects the organization of signals at the cable interface.)
GSC	General Standards Corporation
PCI	Peripheral Component Interconnect
PCIE	PCI Express
PMC	PCI Mezzanine Card
RTS	Request To Send (This is a cable interface signal used for hardware flow control.)

1.3. Definitions

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

Term	Definition
Application	Application means the user mode process, which runs in user space with user mode privileges.
Driver	Driver means the executable providing direct access to the SIO4 hardware.
FASYNC	This refers to the Fast Async serial protocol implementation.
Library	This is a general reference to the Fast Async Protocol Library.
RxD	This refers to the Receive Data signal.
SIO4	This is used as a general reference to any board supported by this driver. This includes both SIO4 and SIO8 model boards. It is also used to refer to revisions of the board that do not include a suffix following the ‘4’, such as PMC-SIO4-RS232.
TxD	This refers to the Transmit Data signal.

1.4. Software Overview

The Fast Async Protocol Library is a statically linked library providing a software interface to the Fast Async version of the SIO4. The Library is provided in source form and must be built before being used. The Library is a software layer that sits between an SIO4 application and the SIO4 API Library. While the other SIO4 Protocol Libraries present a tailored interface to the driver API, this Protocol Library contains functionality which has not been incorporated into the driver. Like the other Protocol Libraries though, this Library exists in parallel with and can be used with the SIO4 API Library.

1.5. Hardware Overview

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

The SIO4 is a four-channel high-speed serial interface I/O board. This board provides for bi-directional serial data transfers between a computer and an external peripheral. Once the data link between the two devices is established, the desired transfers can be performed and will become transparent to the user. The SIO4 board includes two DMA controllers and comes with a maximum of 256K Bytes of FIFO storage, which is 32K per channel direction (32K * 2 * 4). Each DMA controller is capable of transferring data to and from host memory; whereas the FIFO helps maintain continuous data transfer at the cable interface. The FIFO configuration can vary greatly from one SIO4 version to another (i.e., 32K * 2 * 4 to 1K * 2 *1 to none at all). The SIO4 comes with transceivers that are fixed as RS232 or RS485/422, or with transceivers that are configurable. The SIO4 comes in a variety of configurations: SYNC models, Fast Async models, and Zilog models, which are based on two Z16C30 dual USC chips. Later model SIO4 boards support multiple models with the mode being software controlled on a per channel basis. The SIO4 also provides for interrupt generation for various states of the board like Sync Character detection, FIFO empty, FIFO full and DMA complete.

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

1.6. Reference Material

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

- The applicable Fast Async *SIO4/SIO8 User Manual* from General Standards Corporation.

- The applicable *SIO4 API Library Reference Manual* from General Standards Corporation.

- The applicable *SIO4 Driver User Manual* from General Standards Corporation.

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

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

2. The Fast Async Serial Protocol

2.1. Description

The Fast Async serial communications protocol is an asynchronous, byte-oriented serial transmission protocol consisting of a Start Bit, a number of data bits, an optional Parity Bit, and one or two Stop Bits. For successful data transfer the provider and recipient must agree upon the transmission rate, the number of data bits, the use and type of parity, and the minimum length of the Stop Bit period. Refer to Figure 1 below. When no data is being transmitted the line is idle. When idle the line is held in the high or Mark state (a 1 bit). On the receiving side, a transition from high to low signals the beginning of a Start Bit. The line level is then examined four times over the first half of a single bit period. If any of those instances are high, then the high to low transition is ignored as noise and the receiver goes back to waiting for another high to low transition. If all four instances are low (the Space state, which is a 0 bit), then the receiver accepts the transition as a valid Start Bit. Decoding then continues per the byte size, use of parity and Stop Bits. Each of these is examined at the center of the bit period. If parity is used then the Parity Bit is evaluated with the data. If the evaluation fails, then the result is a Parity Error. After the last data bit and optional Parity Bit, the Stop Bit period begins. If the line is low when either Stop Bit is sampled, then the result is a Framing Error.

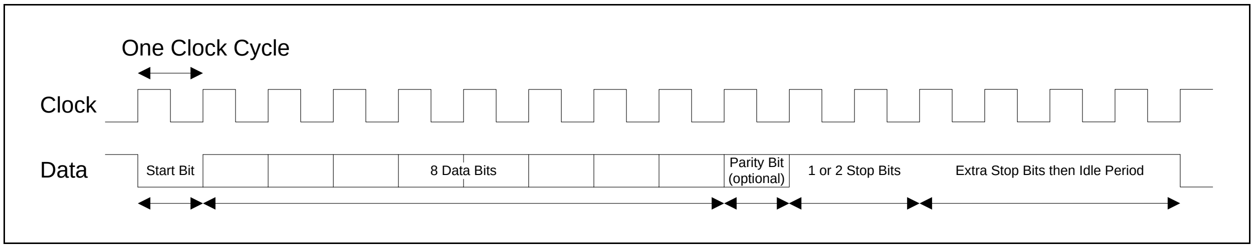

Figure 1 A depiction of a Fast Async data stream.

The minimum signals needed for full-duplex communication are TxD and RxD. Hardware flow control is supported by way of the RTS/CTS signal pair. The Fast Async board supports the use of high-speed RS-422/RS-485 transceivers. The arrangement of the cable signals is configurable as DCE or DTE. The upper baud rate limit is 10Mbps.

2.2. History

The asynchronous serial communications protocol has its origins prior to 1920, back in the days of the early electromechanical teletypewriter. At that time byte sizes were typically five-bits and the Stop Bit period was typically 1.5 times the bit period. Since then, byte size support typically ranges from five to eight bits, and the Stop Bit period may be configurable in fractional increments beginning at just over ½ bit period.

3. Library Interface Files

This section gives general information on the Fast Async Protocol Library interface files.

3.1. Header File

The library's interface is defined via the header file shown below. To use the Fast Async Protocol Library applications must include this header file in their sources. Including this header file pulls in all other pertinent SIO4 specific header files. Therefore, sources may include only this one SIO4 header and make files may reference only this one SIO4 include directory.

File	Location
`sio4_fasync.h`	`…/sio4/include`

3.2. Static Library Files

The executable code for the API defined for the Fast Async Protocol Library is contained in the static library file identified below. Using this library however, requires linking in other SIO4 specific static libraries. For this reason, and for ease of use, it is recommended that application make files link in the SIO4 Main Library instead of the Fast Async static library along with all of its dependencies. The result is that application make files reference only a single SIO4 static library and only a single SIO4 static library path.

Library	File	Location
Fast Async Protocol Library	`sio4_fasync.a`	`…/sio4/lib`
SIO4 Main Library	`sio4_main.a` †	`…/sio4/lib`

† Refer to the SIO4 API Library Reference Manual for clarification when using multiple GSC product types in the same application.

4. Library Interface

The library interface is defined via the header file `sio4_fasync.h`, which is located in the …`/sio4/include/` directory. The library interface is made up of the following content types.

> **NOTE:** All Fast Async specific API content includes a prefix of "`sio4_fasync_`" or "`SIO4_FASYNC_`". The lower-case version is used with functions and data types. The upper-case version is used with macros.

Content	Description
Functions	There are primary, high-level functions and secondary, low-level functions. The high-level functions are those that either deal with overall configuration or are not configuration related. Refer to section 4.1, page 11. The low-level functions deal specifically with accessing individual device configuration features. These are presented with the associated `sio4_fasync_t` structure fields with which each pertains (section 4.3.4, page 40).
Macros	These define registers and data values for the numerous configuration features. Refer to section 4.2, page 35. The configuration option macros are documented with the `sio4_fasync_t` data structure, which is the primary structure used for device configuration. Refer to section 4.3.5, page 40.
Data Types	This includes the enumerations and structures used by the various API functions. Refer to section 4.3, page 37.

4.1. Functions

The interface includes the high-level functions described in the following sections. All function return values reflect the completion status of the requested operation.

Return Values

A return value less than zero always reflects an error condition. The table below summarizes the error status values. For the I/O functions, read and write, non-negative return values reflect the number of bytes transferred between the application and the interface. A positive value equal to the requested transfer size indicates complete success. Return values less than the requested transfer size indicate that the I/O timeout expired. For the other, non-I/O, function calls a return value of zero indicates success.

Return Value	Description
`-1` to `-499`	This is the value "`(-errno)`" (see `errno.h`).

Calling Restrictions

Applying configuration settings in mass should be done using the Set API service (`sio4_fasync_set()`, section 4.1.13, page 26). This is recommended both for application of initial settings and for bulk configuration updates. Thereafter, application should consider using the low-level services. However, many of the low-level services for the hardware-based features have restrictions on when they can be called. Calling restrictions are documented with each API function.

Multi-Thread Safe Operation

All API services are multi-thread safe, except as follows. This includes both the high-level and low-level services.

1. The API Initialize call (`sio4_fasync_init_api()`, section 4.1.5, page 17) is multi-thread safe only after initialization has been successful.

2. The Wait Event call (`sio4_fasync_wait_event()`, section 4.1.19, page 32) is serialized with all other services, but controlled access is released prior to the driver being called.

3. Read requests (`sio4_fasync_read()`, section 4.1.9, page 22) are serialized separately within the Library.

4. Write requests (`sio4_fasync_write()`, section 4.1.21, page 34) are serialized separately within the Library.

4.1.1. sio4_fasync_close()

This function is the entry point to close a connection previously opened to an SIO4. All resources allocated by the Library for the opened device are released as part of the close operation. This includes freeing semaphores, allocated memory and closing access to the SIO4 serial channel.

> **NOTE:** Any attempt to close a connection will fail if the device handle is still in use by another API call.

Prototype

```
int sio4_fasync_close(int fd, sio4_fasync_error_t* error);
```

Argument	**Description**
`fd`	This is a file descriptor obtained from `sio4_fasync_open()` (section 4.1.8, page 20).
`error`	This API uses this optional structure to report detailed error status (section 4.3.3, page 39).

Return Value	**Description**
`0`	The operation succeeded.
`< 0`	An error occurred. See error value description in section 4.1, page 11.

Calling Restrictions

There are no restrictions on calling this function. However, the function will fail if the Library is still busy servicing any other API calls.

4.1.2. sio4_fasync_get()

This is the API's *Get* function, which retrieves the current settings from the SIO4 for each of the referenced structure's fields. This function operates mostly by calling the low-level functions corresponding to each field of the `sio4_fasync_t` structure.

Prototype

```
int sio4_fasync_get(
        int                     fd,
        sio4_fasync_t*          fasync,
        sio4_fasync_error_t*    error);
```

Argument	**Description**
`fd`	This is a file descriptor obtained from `sio4_fasync_open()` (section 4.1.8, page 20).
`fasync`	This required structure is where the current settings are recorded (section 4.3.5, page 40).
`error`	This API uses this optional structure to report detailed error status (section 4.3.3, page 39).

Return Value	**Description**
`0`	The operation succeeded.
`< 0`	An error occurred. See error value description in section 4.1, page 11.

Calling Restrictions

There are no restrictions on calling this function.

4.1.3. sio4_fasync_gpio_rx()

This function reads the cable interface signals GPIO A through GPIO D. The operation is performed by reading the Pin Status Register. The value returned reflects how each cable signal is driven and is independent of how each cable signal is configured, whether it be as GPIO input, output or otherwise.

> **NOTE:** The value obtained is the full 32-bit value read from the register, not just the lower four bits.

Prototype

```
int sio4_fasync_gpio_rx(int fd, u32* get, sio4_fasync_error_t* error);
```

Argument	Description
`fd`	This is a file descriptor obtained from `sio4_fasync_open()` (section 4.1.8, page 20).
`get`	The value obtained is reported via this required pointer.
`error`	This API uses this optional structure to report detailed error status (section 4.3.3, page 39).

Return Value	Description
`0`	The operation succeeded.
`< 0`	An error occurred. See error value description in section 4.1, page 11.

Calling Restrictions

There are no restrictions on calling this function.

4.1.4. sio4_fasync_gpio_tx()

This function updates the state of the cable interface signals configured as GPIO outputs. Those cable signals configured as other than GPIO outputs are unaffected. Bit D0 corresponds to GPIO A, D1 to GPIO B and so on.

> **NOTE:** It is not an error if none of the GPIO signals are configured as GPIO output.

Prototype

```
int sio4_fasync_gpio_tx(int fd, u32 set, sio4_fasync_error_t* error);
```

Argument	Description
`fd`	This is a file descriptor obtained from `sio4_fasync_open()` (section 4.1.8, page 20).
`set`	This is the value to be applied to the GPIO outputs. Only the lower four bits are used. All others are ignored.
`error`	This API uses this optional structure to report detailed error status (section 4.3.3, page 39).

Return Value	Description
`0`	The operation succeeded.
`< 0`	An error occurred. See error value description in section 4.1, page 11.

Calling Restrictions

There are no restrictions on calling this function.

4.1.5. sio4_fasync_init_api()

This function initializes the Fast Async Protocol Library and must be the first call into the Library.

> **NOTE:** This service is not multi-thread safe until after it has successfully initialized the Library.
>
> **NOTE:** This service initializes the Fast Async Protocol Library as well as the SIO4 API Library.
>
> **NOTE:** This function may be called more than once, but only the first successful call initializes the API. Any subsequent call has no effect.

Prototype

```
int sio4_fasync_init_api(sio4_fasync_error_t* error);
```

Argument	**Description**
`error`	This API uses this optional structure to report detailed error status (section 4.3.3, page 39).

Return Value	**Description**
`0`	The operation succeeded.
`< 0`	An error occurred. See error value description in section 4.1, page 11.

Calling Restrictions

This must be the first API call. All other API calls will fail until this function completes successfully.

4.1.6. sio4_fasync_init_data()

This is the API's *Init Data* function, which initializes an `sio4_fasync_t` structure according to the capabilities of the accessed device and some basic caller preferences. This function operates mostly by calling the low-level functions corresponding to each field of the `sio4_fasync_t` structure.

Prototype

```
int sio4_fasync_init_data(
        int                         fd,
        const sio4_fasync_init_t*   init,
        sio4_fasync_t*              fasync,
        sio4_fasync_error_t*        error);
```

Argument	**Description**
`fd`	This is a file descriptor obtained from `sio4_fasync_open()` (section 4.1.8, page 20).
`init`	This required structure (section 4.3.3.1, page 40) provides the basic information needed to initializing the *fasync* argument structure.
`fasync`	This required structure is initialized per the *init* argument and Library defaults. (section 4.3.5, page 40). Most fields can be assigned a value of `-1` to retrieve the current setting rather than to apply a setting.
`error`	This API uses this optional structure to report detailed error status (section 4.3.3, page 39).

Return Value	**Description**
`0`	The operation succeeded.
`< 0`	An error occurred. See error value description in section 4.1, page 11.

Calling Restrictions

There are no restrictions on calling this function.

4.1.7. sio4_fasync_ioctl()

This function is the entry point to performing IOCTL operations on the device. Refer to the SIO4 API Library Reference Manual for complete information on the supported set of IOCTL services. There are no Fast Async specific IOCTL services.

> **NOTE:** The Library performs no validation of which IOCTL services are or are not supported by the accessed device. Any validation performed is done by the driver. Applications should avoid requesting services which are not specifically known to be supported by the accessed device. Doing so could result in unexpected behavior due to the driver's limited awareness of Fast Async device features.

Prototype

```
int sio4_fasync_ioctl(
        int                     fd,
        int                     request,
        void*                   arg,
        sio4_fasync_error_t*    error);
```

Argument	Description
`fd`	This is a file descriptor obtained from `sio4_fasync_open()` (section 4.1.8, page 20).
`request`	This is an IOCTL macro contained in `sio4.h` or `sio4_usc.h`. Most services which take an argument type of `s32*` will accept a value of `-1` to retrieve the feature's current setting.
`arg`	This is the argument type required for the above referenced IOCTL service.
`error`	This API uses this optional structure to report detailed error status (section 4.3.3, page 39).

Return Value	Description
`0`	The operation succeeded.
`< 0`	An error occurred. See error value description in section 4.1, page 11.

Calling Restrictions

There are no restrictions on calling this function. However, applications should not call this function using the `SIO4_IOCTL_WAIT_EVENT` IOCTL service. If this is done then it will prevent use of most other API services until this call returns. This is because all API services are serialized and because this is a blocking service, which will not return till after the specified event criteria occurs or the timeout expires.

4.1.8. sio4_fasync_open()

This function is the entry point to opening a connection to an SIO4 serial channel. The handle returned by this call is used for all subsequent access to the specified channel.

> **NOTE:** The file descriptor returned can be used with all services in both this Library and the SIO4 API Library. The reverse is not supported.
>
> **NOTE**: If the value of the *index* argument is `-1`, then the function opens the file `/proc/sio4`. In this case the `share` argument is ignored. In addition, the file descriptor is valid only with the read and close Library services.

Prototype

```
int sio4_fasync_open(
        int                     index,
        int                     share,
        int*                    fd,
        sio4_fasync_error_t*    error);
```

Argument	**Description**
`index`	This is the zero-based index of the SIO4 serial channel to access.
`share`	Open the device in Shared Access Mode? If non-zero the device is opened in Shared Access Mode (see below). If zero the device is opened in Exclusive Access Mode (see below).
`fd`	The device handle is returned here. If this argument is NULL, then an error is reported. Values returned are as follows. <table><tr><th>Value</th><th>Description</th></tr><tr><td><code>>= 0</code></td><td>This is the handle to use to access the device in subsequent calls.</td></tr><tr><td><code>-1</code></td><td>There was an error. The device is not accessible.</td></tr></table>
`error`	This API uses this optional structure to report detailed error status (section 4.3.3, page 39).

Return Value	**Description**
`0`	The operation succeeded.
`< 0`	An error occurred. See error value description in section 4.1, page 11.

Calling Restrictions

There are no restrictions on calling this function.

4.1.8.1. Access Modes

The value of the `share` argument determines the device access mode, as follows.

Shared Access Mode:

If the `share` argument is non-zero the device in opened in Shared Access Mode. The first such open request will succeed and return with the device in an initialized state. Subsequent such open requests will also succeed, but will not alter the device state. Once opened in Shared Access Mode, device access remains in this mode until all Shared Access Mode open requests release the device with a corresponding close request.

Exclusive Access Mode:

If the `share` argument is zero the device in opened in Exclusive Access Mode. In this mode, only one application at a time can access the device. The first such open request will succeed and return with the device in an initialized state. Subsequent open requests, regardless of the `share` argument value, will fail until the device is released with a corresponding close request.

4.1.9. sio4_fasync_read()

This function requests that a buffer of data be read from the serial channel. The request will return either when it has been fulfilled or the read timeout expires, whichever occurs first. This is a blocking call. Read requests are serialized separately from all other API services.

Prototype

```
int sio4_fasync_read(
        int                     fd,
        void*                   dst,
        size_t                  bytes,
        sio4_fasync_error_t*    error);
```

Argument	Description
`fd`	This is a file descriptor obtained from `sio4_fasync_open()` (section 4.1.8, page 20).
`dst`	The data read is placed here.
`bytes`	Read up to this number of bytes.
`error`	This API uses this optional structure to report detailed error status (section 4.3.3, page 39).

Return Value	Description
`0` to `bytes`	The operation succeeded. This is the number of bytes placed in the buffer. A value less than `bytes` generally indicates that the I/O timeout period lapsed before the entire request could be satisfied.
`< 0`	An error occurred. See error value description in section 4.1, page 11.

Calling Restrictions

There are no restrictions on calling this function. However, because read requests are serialized separately from all other API services, applications must exercise care with regard to how all other services might impact ongoing read operations.

4.1.10. sio4_fasync_reset()

This function performs a reset of the serial channel. This includes all settings, both hardware and software based. However, this does not reprogram the oscillator and thus it does not affect the configured bitrate. The reset operation takes place immediately and does not wait for the transmitter or receiver to finish a byte in progress.

> **NOTE:** If an application must wait for transmission or reception to complete before performing a reset, then refer to section 5.6, page 103.

Prototype

```
int sio4_fasync_reset(int fd, sio4_fasync_error_t* error);
```

Argument	**Description**
`fd`	This is a file descriptor obtained from `sio4_fasync_open()` (section 4.1.8, page 20).
`error`	This API uses this optional structure to report detailed error status (section 4.3.3, page 39).

Return Value	**Description**
`0`	The operation succeeded.
`< 0`	An error occurred. See error value description in section 4.1, page 11.

Calling Restrictions

There are no restrictions on calling this function.

4.1.11. sio4_fasync_rx_fifo_reset()

This function clears the Rx FIFO of its content and clears the overflow and underflow status, if either or both are set. This operation has no effect on the receiver's operation if it is in the process of receiving a byte.

Prototype

```
int sio4_fasync_rx_fifo_reset(int fd, sio4_fasync_error_t* error);
```

Argument	Description
`fd`	This is a file descriptor obtained from `sio4_fasync_open()` (section 4.1.8, page 20).
`error`	This API uses this optional structure to report detailed error status (section 4.3.3, page 39).

Return Value	Description
`0`	The operation succeeded.
`< 0`	An error occurred. See error value description in section 4.1, page 11.

Calling Restrictions

There are no restrictions on calling this function.

4.1.12. sio4_fasync_rx_io_abort()

This function aborts a read request, if one is active.

Prototype

```
int sio4_fasync_rx_io_abort(
        int                     fd,
        s32*                    get,
        sio4_fasync_error_t*    error);
```

Argument	Description
`fd`	This is a file descriptor obtained from `sio4_fasync_open()` (section 4.1.8, page 20).
`get`	The results of the operation are recorded here, if non-NULL. See result options below.
`error`	This API uses this optional structure to report detailed error status (section 4.3.3, page 39).

Return Value	Description
`0`	The operation succeeded.
`< 0`	An error occurred. See error value description in section 4.1, page 11.

Results

Upon success, the function returns one of the below values.

Value	Description
`SIO4_FASYNC_IO_ABORT_NO`	A read request was not aborted.
`SIO4_FASYNC_IO_ABORT_YES`	A read request was aborted.

Calling Restrictions

There are no restrictions on calling this function.

4.1.13. sio4_fasync_set()

This is the API's *Set* function, which configures an SIO4 channel according to the settings of the referenced `sio4_fasync_t` structure. All fields are validated before any settings are applied. This function operates mostly by calling the low-level functions corresponding to each field of the `sio4_fasync_t` structure.

> **NOTE:** Before calling this function, the structure should, at minimum, be initialized by calling the `sio4_fasync_init_data()` function (section 4.1.6, page 14).
>
> **NOTE:** The cable transceivers, the transmitter and the receiver are disabled while the configuration settings are being applied.

Prototype

```
int sio4_fasync_set(
        int                     fd,
        sio4_fasync_t*          fasync,
        sio4_fasync_error_t*    error);
```

Argument	Description
`fd`	This is a file descriptor obtained from `sio4_fasync_open()` (section 4.1.8, page 20).
`fasync`	This required structure contains the settings to be applied (section 4.3.5, page 40). The value in fields which are read-only are ignored and are overwritten by the current setting. Fields with a value of `-1` also retrieve the current setting.
`error`	This API uses this optional structure to report detailed error status (section 4.3.3, page 39).

Return Value	Description
`0`	The operation succeeded.
`< 0`	An error occurred. See error value description in section 4.1, page 11.

Calling Restrictions

There are no restrictions on calling this function.

4.1.14. sio4_fasync_show()

This is the API's *Show* function, which reports the content of the referenced `sio4_fasync_t` structure to the specified file. This is provided to assist debugging efforts. This function operates mostly by calling the low-level functions corresponding to each field of the `sio4_fasync_t` structure.

Prototype

```
int sio4_fasync_show(
        int                     fd,
        const sio4_fasync_t*    fasync,
        FILE*                   file,
        sio4_fasync_error_t*    error);
```

Argument	Description
`fd`	This is a file descriptor obtained from `sio4_fasync_open()` (section 4.1.8, page 20).
`fasync`	This required structure holds the content to be reported (section 4.3.5, page 40).
`file`	This is a file pointer to which the output is sent. If this is NULL, then no output is generated. If this is `stdout`, then the output is sent to the terminal window. Output will otherwise be written to the referenced file.
`error`	This API uses this optional structure to report detailed error status (section 4.3.3, page 39).

Return Value	Description
`0`	The operation succeeded.
`< 0`	An error occurred. See error value description in section 4.1, page 11.

Calling Restrictions

There are no restrictions on calling this function.

4.1.15. sio4_fasync_tx_fifo_reset()

This function clears the Tx FIFO of its content and clears the overflow status, if it is set. This operation has no effect on the transmitter's operation if it is in the process of transmitting a byte.

Prototype

```
int sio4_fasync_tx_fifo_reset(int fd, sio4_fasync_error_t* error);
```

Argument	Description
`fd`	This is a file descriptor obtained from `sio4_fasync_open()` (section 4.1.8, page 20).
`error`	This API uses this optional structure to report detailed error status (section 4.3.3, page 39).

Return Value	Description
`0`	The operation succeeded.
`< 0`	An error occurred. See error value description in section 4.1, page 11.

Calling Restrictions

There are no restrictions on calling this function.

4.1.16. sio4_fasync_tx_io_abort()

This function aborts a write request, if one is active.

Prototype

```
int sio4_fasync_tx_io_abort(
        int                     fd,
        s32*                    get,
        sio4_fasync_error_t*    error);
```

Argument	Description
`fd`	This is a file descriptor obtained from `sio4_fasync_open()` (section 4.1.8, page 20).
`get`	The results of the operation are recorded here, if non-NULL. See result options below.
`error`	This API uses this optional structure to report detailed error status (section 4.3.3, page 39).

Return Value	Description
`0`	The operation succeeded.
`< 0`	An error occurred. See error value description in section 4.1, page 11.

Results

Upon success, the function returns one of the below values.

Value	Description
`SIO4_FASYNC_IO_ABORT_NO`	A write request was not aborted.
`SIO4_FASYNC_IO_ABORT_YES`	A write request was aborted.

Calling Restrictions

There are no restrictions on calling this function.

4.1.17. sio4_fasync_verify()

This is the API's *Verify* function, which verifies the values in each field of the referenced `sio4_fasync_t` structure. All fields are validated. This function operates mostly by calling the low-level functions corresponding to each field of the `sio4_fasync_t` structure.

> **NOTE:** Before calling this function, the structure should, at minimum, be initialized by calling the `sio4_fasync_init_data()` function (section 4.1.6, page 14).

Prototype

```
int sio4_fasync_verify(
        int                     fd,
        const sio4_fasync_t*    fasync,
        sio4_fasync_error_t*    error);
```

Argument	**Description**
`fd`	This is a file descriptor obtained from `sio4_fasync_open()` (section 4.1.8, page 20).
`fasync`	This required structure contains the settings to be verified (section 4.3.5, page 40). The value in fields which are read-only are ignored and are overwritten by the current setting. Fields with a value of `-1` also retrieve the current setting.
`error`	This API uses this optional structure to report detailed error status (section 4.3.3, page 39).

Return Value	**Description**
`0`	The operation succeeded.
`< 0`	An error occurred. See error value description in section 4.1, page 11.

Calling Restrictions

There are no restrictions on calling this function.

4.1.18. sio4_fasync_wait_cancel()

This function cancels all active Wait Event requests meeting any of the specified criteria. Cancelation is based on any matching criteria as described below. The cancel is performed by resuming the blocked thread. All active Wait Event requests are checked.

Prototype

```
int sio4_fasync_wait_cancel(
        int                     fd,
        gsc_wait_t*             wait,
        sio4_fasync_error_t*    error);
```

Argument	Description
`fd`	This is a file descriptor obtained from `sio4_fasync_open()` (section 4.1.8, page 20).
`wait`	This required structure specifies the criteria for which any matching Wait Event requests are to be canceled (section 4.3.1, page 37).
`error`	This API uses this optional structure to report detailed error status (section 4.3.3, page 39).

Return Value	Description
`0`	The operation succeeded.
`< 0`	An error occurred. See error value description in section 4.1, page 11.

Wait Structure Usage

> **NOTE**: The *alt* field is the only field validated by the Fast Async Protocol Library. Any other validation performed is carried out by the SIO4 API Library and the SIO4 device driver.

The wait structure is used as follows.

Field	Description
`flags`	This field is not used and should be zero upon return.
`main`	This field specifies any of the `GSC_WAIT_MAIN_*` flags to be checked against active Wait Event requests. Refer to section 4.3.1.2, page 37.
`gsc`	This field specifies any of the `SIO4_FASYNC_WAIT_GSC_*` flags to be checked against active Wait Event requests. Refer to section 4.3.1.3, page 38.
`alt`	This field must be set to zero.
`io`	This field specifies any of the `SIO4_FASYNC_WAIT_IO_*` flags to be checked against active Wait Event requests. Refer to section 4.3.1.5, page 38.
`timeout_ms`	This field is ignored and should be zero upon return.
`count`	This is ignored on input and, upon return, is set to the number of Wait Event requests that were cancelled.

Calling Restrictions

There are no restrictions on calling this function.

4.1.19. sio4_fasync_wait_event()

This function suspends the calling thread while waiting for any of the specified events to occur, or until a timeout lapses, whichever occurs first. Any number or combination of events may be specified, but resumption occurs based on the first event that matches any of the specified criteria. All fields must be valid and at least one criterion must be specified. There is no restriction on the number of simultaneously active Wait Event requests.

Prototype

```
int sio4_fasync_wait_event(
        int                     fd,
        gsc_wait_t*             wait,
        sio4_fasync_error_t*    error);
```

Argument	Description
`fd`	This is a file descriptor obtained from `sio4_fasync_open()` (section 4.1.8, page 20).
`wait`	This required structure specifies the criteria to wait on (section 4.3.1, page 37).
`error`	This API uses this optional structure to report detailed error status (section 4.3.3, page 39).

Return Value	Description
`0`	The operation succeeded.
`< 0`	An error occurred. See error value description in section 4.1, page 11.

Wait Structure Usage

> **NOTE**: The *alt* field is the only field validated by the Fast Async Protocol Library. Any other validation performed is carried out by the SIO4 API Library and the SIO4 device driver.

The wait structure is used as follows.

Field	Description
`flags`	On input, this must be zero. Upon return, one of the `GSC_WAIT_FLAG_*` flags will be set to indicate the reason that the thread was resumed. Refer to section 4.3.1.1, page 37.
`main`	This field specifies any of the `GSC_WAIT_MAIN_*` criteria to wait for. Refer to section 4.3.1.2, page 37.
`gsc`	This field specifies any of the `SIO4_FASYNC_WAIT_GSC_*` criteria to wait for. Refer to section 4.3.1.3, page 38.
`alt`	This field must be set to zero.
`io`	This field specifies any of the `SIO4_FASYNC_WAIT_IO_*` criteria to wait for. Refer to section 4.3.1.5, page 38.
`timeout_ms`	This specified the maximum amount of time, in milliseconds, that the thread is to wait for any of the referenced events. A value of zero means do not timeout at all. If non-zero, then upon return the value will be the approximate amount of time actually waited.
`count`	This is unused by Wait Event operations and must be zero.

Calling Restrictions

There are no restrictions on calling this function.

> **NOTE:** This function is serialized with most other API calls. However, the semaphore used to serialize access is released before the Library passes control off the SIO4 API Library.

4.1.20. sio4_fasync_wait_status()

This function retrieves a count of threads waiting on any of the specified criteria. A match is made when a waiting thread's wait criteria matches any of the criteria specified in the structure passed to this service.

Prototype

```
int sio4_fasync_wait_status(
        int                     fd,
        gsc_wait_t*             wait,
        sio4_fasync_error_t*    error);
```

Argument	**Description**
`fd`	This is a file descriptor obtained from `sio4_fasync_open()` (section 4.1.8, page 20).
`wait`	This required structure specifies the criteria to wait on (section 4.3.1, page 37).
`error`	This API uses this optional structure to report detailed error status (section 4.3.3, page 39).

Return Value	**Description**
`0`	The operation succeeded.
`< 0`	An error occurred. See error value description in section 4.1, page 11.

Wait Structure Usage

> **NOTE**: The *alt* field is the only field validated by the Fast Async Protocol Library. Any other validation performed is carried out by the SIO4 API Library and the SIO4 device driver.

The wait structure is used as follows.

Field	**Description**
`flags`	This is ignored and should be zero upon return.
`main`	This field specifies the `GSC_WAIT_MAIN_*` criteria whose matching Wait Event requests are to be counted. Refer to section 4.3.1.2, page 37.
`gsc`	This field specifies any of the `SIO4_FASYNC_WAIT_GSC_*` criteria whose matching Wait Event requests are to be counted. Refer to section 4.3.1.3, page 38.
`alt`	This is ignored and should be zero upon return.
`io`	This field specifies any of the `SIO4_FASYNC_WAIT_IO_*` criteria whose matching Wait Event requests are to be counted. Refer to section 4.3.1.5, page 38.
`timeout_ms`	This is ignored and should be zero upon return.
`count`	This is ignored on input. Upon return this indicates the number of Wait Event requests that met any of the specified criteria.

Calling Restrictions

There are no restrictions on calling this function.

4.1.21. sio4_fasync_write()

This function requests that a buffer of data be written to the serial channel. The request will return either when it has been fulfilled or the write timeout expires, whichever occurs first. This is a blocking call.

Prototype

```
int sio4_fasync_write(
        int                     fd,
        const void*             src,
        size_t                  bytes,
        sio4_fasync_error_t*    error);
```

Argument	Description
`fd`	This is a file descriptor obtained from `sio4_fasync_open()` (section 4.1.8, page 20).
`src`	This is the source for the data to write.
`bytes`	This is the number of bytes to write.
`error`	This API uses this optional structure to report detailed error status (section 4.3.3, page 39).

Return Value	Description
`0` to `bytes`	The operation succeeded. This is the number of bytes written from the buffer. A value less than `bytes` generally indicates that the I/O timeout period lapsed before the entire request could be satisfied.
`< 0`	An error occurred. See error value description in section 4.1, page 11.

Calling Restrictions

There are no restrictions on calling this function. However, because write requests are serialized separately from all other API services, applications must exercise care with regard to how all other services might impact ongoing write operations.

4.1.22. Low Level Functions

The low-level functions provide access to the board features reflected by the individual fields of the `sio4_fasync_t` structure (section 4.3.5, page 40). This structure is used to access all of the board configuration features that are part of the API. For each structure field there is a corresponding board feature and an associated low-level function. When calling the high-level functions that use the `sio4_fasync_t` structure, the high-level functions perform their work by calling the low-level functions for each respective structure field. This is especially useful for activities such as structure initialization and board configuration. Following high level configuration of the board though, there are times when an application may need to access features represented by only a subset of the `sio4_fasync_t` structure fields. This is where an application can make use of the low-level functions. All of the low-level functions follow the prototype pattern shown below. The function naming convention includes the prefix "`sio4_fasync_t_`" followed by the C style field name, but with the periods ("`.`") replaced by underscores ("`_`").

Prototype

```
void sio4_fasync_t_field_name(
    int                     fd,
    s32*                    arg,
    sio4_fasync_action_t    action,
    FILE*                   file,
    sio4_fasync_error_t*    error);
```

Argument	**Description**
`fd`	This is a file descriptor obtained from `sio4_fasync_open()` (section 4.1.8, page 20).
`arg`	This required pointer refers to the feature specific value passed to or returned from the function.
`action`	This identifies the specific action to be carried out in regards to the referenced feature (section 4.3.2, page 39).
`file`	This is a file pointer used by the Show service (section 4.1.14, page 27) to where the output is directed. If this is NULL, then no output is generated. For output to a terminal screen `stdout` should be used. Otherwise, the output is written to the specified file.
`error`	This API uses this optional structure to report detailed error status (section 4.3.3, page 39).

Examples

For simplicity's sake a low-level function name can easily be derived given any field name, as shown in the below examples. The individual low-level function names are identified with the corresponding structure fields beginning in section 4.3.5, page 40.

Field	**Function**
`sio4_fasync_t.cable.loopback`	`sio4_fasync_t_cable_loopback()`
`sio4_fasync_t.rx.enable`	`sio4_fasync_t_rx_enable()`
`sio4_fasync_t.tx.io.timeout`	`sio4_fasync_t_tx_io_timeout()`

4.2. Macros

The API includes the following macros. The Fast Async specific macros are defined in `sio4_fasync.h`. The header for any other macros is identified below with the macro descriptions.

4.2.1. Registers

The following gives the complete set of Fast Async registers.

4.2.1.1. Firmware Registers

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

> **NOTE**: Refer to the output of the "id" sample application (…`/id/`) for a complete list of the registers supported by the device being accessed.

Macros	**Description**
`SIO4_FASYNC_ACSR`	Asynchronous Control/Status Register (ACSR)
`SIO4_FASYNC_BCR`	Board Control Register (BCR)
`SIO4_FASYNC_BSR`	Board Status Register (BSR)
`SIO4_FASYNC_CSR`	Control/Status Register (CSR)
`SIO4_FASYNC_FCR`	FIFO Count Register (FCR)
`SIO4_FASYNC_FDR`	FIFO Data Register (FDR)
`SIO4_FASYNC_FR`	Features Register (FR)
`SIO4_FASYNC_FRR`	Firmware Revision Register (FRR)
`SIO4_FASYNC_FSR`	FIFO Size Register (FSR)
`SIO4_FASYNC_FTR`	Firmware Type Register (FTR)
`SIO4_FASYNC_ICR`	Interrupt Control Register (ICR)
`SIO4_FASYNC_IELR`	Interrupt Edge/Level Register (IELR)
`SIO4_FASYNC_IHLR`	Interrupt High/Low Register (IHLR)
`SIO4_FASYNC_ISR`	Interrupt Status Register (ISR)
`SIO4_FASYNC_POCSR`	Programmable Oscillator Control/Status Register (POCSR)
`SIO4_FASYNC_PORAR`	Programmable Oscillator RAM Address Register (PORAR)
`SIO4_FASYNC_PORD2R`	Programmable Oscillator RAM Data 2 Register (PORD2R)
`SIO4_FASYNC_PORDR`	Programmable Oscillator RAM Data Register (PORDR)
`SIO4_FASYNC_PSRCR`	Pin Source Register (PSRCR)
`SIO4_FASYNC_PSTSR`	Pin Ststus Register (PSTSR)
`SIO4_FASYNC_RAR`	Rx Almost Register (RAR)
`SIO4_FASYNC_TAR`	Tx Almost Register (TAR)
`SIO4_FASYNC_TGR`	Tx Gap Register (TGR)

4.2.1.2. PCI Configuration Registers

Access to the PCI registers is seldom required so these registers are not listed here. For the complete list of the PCI register identifiers refer to header file `gsc_pci9056.h`, which is automatically included via `sio4_fasync_api.h`.

4.2.1.3. PLX Feature Set Registers

Access to the PLX registers is seldom required so these registers are not listed here. For the complete list of the PLX register identifiers refer to header file `gsc_pci9056.h`, which is automatically included via `sio4_fasync_api.h`.

4.2.2. Other Macros

All other macros are defined along with the structure fields and/or functions with which they are used.

4.3. Data Types

4.3.1. gsc_wait_t

This structure is used by the Library to process Wait Event, Wait Cancel and Wait Status requests. The structure is defined in the header file `gsc_common.h`.

```
typedef struct
{
    u32 flags;
    u32 main;
    u32 gsc;
    u32 alt;
    u32 io;
    u32 timeout_ms;
    u32 count;
} gsc_wait_t;
```

4.3.1.1. gsc_wait_t.flags

This field is used by the Library to report the results of Wait Event requests. On input, this field must be zero for Wait Event requests and is ignored for Wait Cancel and Wait Status requests. Upon return, this field will be one of the values from the table below for Wait Event requests and zero for Wait Cancel and Wait Status requests.

Valid Values

This field reports the reason that Wait Event requests were resumed. The results will include one of the options given below.

Value	Description
`GSC_WAIT_FLAG_CANCEL`	The wait was cancelled.
`GSC_WAIT_FLAG_DONE`	The wait was completed successfully. The event which completed the request is reported through one of the Wait Event macros given below.
`GSC_WAIT_FLAG_TIMEOUT`	The timeout period expired.

4.3.1.2. gsc_wait_t.main

This field is used to specify which of the below listed event types are of interest.

Valid Values

The main events are as follows.

Value	Description
`GSC_WAIT_MAIN_ALL`	This refers to all of the below event types.
`GSC_WAIT_MAIN_DMA0`	This refers to a DMA Done interrupt from the first DMA engine.
`GSC_WAIT_MAIN_DMA1`	This refers to a DMA Done interrupt from the second DMA engine.
`GSC_WAIT_MAIN_GSC`	This refers to any and all GSC based firmware interrupts.
`GSC_WAIT_MAIN_OTHER`	This refers to other interrupts known to the PLX PCI bridge chip, but which are not itemized here.
`GSC_WAIT_MAIN_PCI`	This refers to any and all PCI based device interrupts.
`GSC_WAIT_MAIN_SPURIOUS`	This refers to interrupts which are supported by the driver, but not used by the SIO4.
`GSC_WAIT_MAIN_UNKNOWN`	This refers to interrupts the driver could not identify.

4.3.1.3. gsc_wait_t.gsc

This field is used to specify GSC firmware-based interrupts. It is the application’s responsibility to enable these interrupts in preparation for Wait Event requests.

Valid Values

The firmware interrupt events are as follows.

Value	Description
`SIO4_FASYNC_WAIT_GSC_ALL`	This refers to all of the below event types.
`SIO4_FASYNC_WAIT_GSC_RX_FIFO_AF`	This refers to the Rx FIFO becoming Almost Full.
`SIO4_FASYNC_WAIT_GSC_RX_FIFO_E`	This refers to the Rx FIFO becoming empty.
`SIO4_FASYNC_WAIT_GSC_RX_FIFO_F`	This refers to the Rx FIFO becoming full.
`SIO4_FASYNC_WAIT_GSC_RX_FRAM_ERR`	This refers to an Rx FIFO frame error.
`SIO4_FASYNC_WAIT_GSC_RX_PAR_ERR`	This refers to an Rx FIFO parity error.
`SIO4_FASYNC_WAIT_GSC_TX_FIFO_AE`	This refers to the Tx FIFO becoming Almost Empty.
`SIO4_FASYNC_WAIT_GSC_TX_FIFO_E`	This refers to the Tx FIFO becoming empty.
`SIO4_FASYNC_WAIT_GSC_TX_FIFO_F`	This refers to the Tx FIFO becoming full.

4.3.1.4. gsc_wait_t.alt

This field is unused by the Fast Async Protocol Library. It should be zero for Wait Event requests and is ignored for Wait Cancel and Wait Status requests.

4.3.1.5. gsc_wait_t.io

This field is used to specify I/O based events.

Valid Values

The I/O events are as follows.

Value	Description
`SIO4_FASYNC_WAIT_IO_ALL`	This refers to all of the below event types.
`SIO4_FASYNC_WAIT_IO_RX_ABORT`	This refers to a read request being aborted (section 4.1.9, page 24).
`SIO4_FASYNC_WAIT_IO_RX_DONE`	This refers to read request being completed successfully.
`SIO4_FASYNC_WAIT_IO_RX_ERROR`	This refers to a read request being ended due to an error.
`SIO4_FASYNC_WAIT_IO_RX_TIMEOUT`	This refers to a read request being ended due to a timeout.
`SIO4_FASYNC_WAIT_IO_TX_ABORT`	This refers to a write request being aborted (section 4.1.21, page 28).

`SIO4_FASYNC_WAIT_IO_TX_DONE`	This refers to write request being completed successfully.
`SIO4_FASYNC_WAIT_IO_TX_ERROR`	This refers to a write request being ended due to an error.
`SIO4_FASYNC_WAIT_IO_TX_TIMEOUT`	This refers to a write request being ended due to a timeout.

4.3.1.6. gsc_wait_t.timeout_ms

This field is used for Wait Event requests only. It is used to specify the upper limit on the amount of time, in milliseconds, the application is willing to wait for the specified request. This field is ignored for Wait Cancel and Wait Status requests and should be zero upon return.

4.3.1.7. gsc_wait_t.count

This field is used for Wait Cancel and Wait Status requests. Upon return, this field indicates either the number waiting threads that were cancelled or the number of waiting threads that met any of the specified status criteria. This field is ignored for Wait Event requests and should be zero upon return.

4.3.2. sio4_fasync_action_t

This enumeration identifies the specific action desired when a low-level function is called.

```
typedef enum
{
   SIO4_FASYNC_ACTION_GET,
   SIO4_FASYNC_ACTION_INIT,
   SIO4_FASYNC_ACTION_SET,
   SIO4_FASYNC_ACTION_SHOW,
   SIO4_FASYNC_ACTION_VERIFY
} sio4_fasync_action_t;
```

Value	Description
`SIO4_FASYNC_ACTION_GET`	This requests the current setting.
`SIO4_FASYNC_ACTION_INIT`	This requests the default setting.
`SIO4_FASYNC_ACTION_SET`	This requests that the supplied setting be applied by the device.
`SIO4_FASYNC_ACTION_SHOW`	This requests that the supplied setting be displayed to the screen. †
`SIO4_FASYNC_ACTION_VERIFY`	This requests that the supplied setting be validated.

† The output may be directed to a file or disabled.

4.3.3. sio4_fasync_error_t

This structure is used by the Library to provide detailed error information when an error is encountered. Applications are not required to use this structure. When not used, applications pass in a NULL pointer. When used, the `ret` field must be set to zero before calling the API. If multiple errors are encountered, this structure will always refer to the very first that appeared. If an application makes multiple, sequential calls to the API, then the application can choose to set the `ret` field to zero only before the first API call. In so doing, the API will record the first error only as it will not overwrite the structure if the `ret` field is non-zero.

```
typedef struct
{
     int            ret;
     char           msg[128];
     const char*    name;
     const char*    file;
     const char*    function;
     int            line;
} sio4_fasync_error_t;
```

4.3.3.1. sio4_fasync_error_t.ret

The is the function return value as described in section 4.1, page 11.

4.3.3.2. sio4_fasync_error_t.msg

This gives a brief description of the error. This will be empty if no error is encountered.

4.3.3.3. sio4_fasync_error_t.name

This gives the name of the API component called or referenced by the function called. This may be a function or field name and will identify either the function called by the application or the low-level function (or field) if called by the API. This will be NULL when no error is encountered.

4.3.3.4. sio4_fasync_error_t.file

This will be the API source file name where the error appeared or NULL if no error occurred.

4.3.3.5. sio4_fasync_error_t.function

This will be the API function name where the error appeared or NULL if no error occurred.

4.3.3.6. sio4_fasync_error_t.line

This will be the API source file line number where the error appeared or zero if no error occurred.

4.3.4. sio4_fasync_init_t

This structure is used by the application to specify the data needed by the API when generating a starting, default configuration. The structure is initialized by the application and passed to the `sio4_fasync_init_data()` function (section 4.1.6, page 18).

```
typedef struct
{
    S32  bitrate;
} sio4_fasync_init_t;
```

4.3.4.1. sio4_fasync_init_t.bitrate

This field specifies the desired bitrate for the transmitter and the receiver. The Library uses this value to derive the target frequency for the programmable oscillator. Valid values are the inclusive range from one to 10,000,000.

4.3.5. sio4_fasync_t

This structure contains all of the fields needed to configure an SIO4 Fast Async serial channel. Most fields are user configurable, though a few are read-only and are provided for informational purposes only. All structure fields are described in the following subsections. Each field description includes all applicable macros as well as the associated low-level API function. All fields are filled in when the `sio4_fasync_init_data()` function is called (section 4.1.6, page 17). Application modifications must be made prior to calling `sio4_fasync_set()` (section 4.1.13, page 25).

```
typedef struct
{
    s32            bitrate;  // read-only    section 4.3.5.1, page 43
    s32            board_leds;               section 4.3.5.2, page 44
```

```
s32            chan_leds;                section 4.3.5.3, page 45
s32            fw_type;  // read-only    section 4.3.5.4, page 46
s32            osc_program;              section 4.3.5.5, page 47
s32            stop_bits;                section 4.3.5.6, page 48

struct         // cable
{
     s32       enable;                   section 4.3.5.7, page 49
     s32       gpio_a;                   section 4.3.5.8, page 50
     s32       gpio_b;                   section 4.3.5.9, page 51
     s32       gpio_c;                   section 4.3.5.10, page 52
     s32       gpio_d;                   section 4.3.5.11, page 53
     s32       loopback;                 section 4.3.5.12, page 54
     s32       mode;                     section 4.3.5.13, page 55
     s32       protocol;                 section 4.3.5.14, page 56
     s32       term;                     section 4.3.5.15, page 57
} cable;

struct         // irq
{
     s32       direction;                section 4.3.5.16, page 58
     s32       enable;                   section 4.3.5.17, page 59
} irq;

struct         // parity
{
     s32       enable;                   section 4.3.5.18, page 60
     s32       type;                     section 4.3.5.19, page 61
} parity;

struct         // tx
{
     s32       cts;                      section 4.3.5.20, page 62
     s32       data;                     section 4.3.5.21, page 63
     s32       enable;                   section 4.3.5.22, page 64
     s32       gap;                      section 4.3.5.23, page 65
     s32       status;   // read-only    section 4.3.5.24, page 66

     struct    // tx.fifo
     {
          s32  ae;                       section 4.3.5.25, page 67
          s32  af;                       section 4.3.5.26, page 68
          s32  bytes;    // read-only    section 4.3.5.27, page 69
          s32  empty_cfg;                section 4.3.5.28, page 70
          s32  overflow;                 section 4.3.5.29, page 71
          s32  size;     // read-only    section 4.3.5.30, page 72
          s32  status;   // read-only    section 4.3.5.31, page 73
     } fifo;

     struct    // tx.io
     {
          s32  dma_threshold;            section 4.3.5.32, page 74
          s32  mode;                     section 4.3.5.33, page 75
          s32  overflow;                 section 4.3.5.34, page 76
```

```
            s32 pio_threshold;        section 4.3.5.35, page 77
            s32 timeout;              section 4.3.5.36, page 78
        } io;
    } tx;

    struct      // rx
    {
        s32     enable;               section 4.3.5.37, page 79
        s32     frame_err_cfg;        section 4.3.5.38, page 80
        s32     parity_err_cfg;       section 4.3.5.39, page 81
        s32     rts;                  section 4.3.5.40, page 82
        s32     sb2_err_cfg;          section 4.3.5.41, page 83
        s32     status; // read-only  section 4.3.5.42, page 84

        struct  // rx.fifo
        {
            s32 ae;                   section 4.3.5.43, page 85
            s32 af;                   section 4.3.5.44, page 86
            s32 bytes;  // read-only  section 4.3.5.45, page 87
            s32 full_cfg;             section 4.3.5.46, page 88
            s32 overflow;             section 4.3.5.47, page 89
            s32 size;   // read-only  section 4.3.5.48, page 90
            s32 status; // read-only  section 4.3.5.49, page 91
            s32 underflow;            section 4.3.5.50, page 92
        } fifo;

        struct  // rx.io
        {
            s32 dma_threshold;        section 4.3.5.51, page 93
            s32 mode;                 section 4.3.5.52, page 94
            s32 overflow;             section 4.3.5.53, page 95
            s32 pio_threshold;        section 4.3.5.54, page 96
            s32 timeout;              section 4.3.5.55, page 97
            s32 underflow;            section 4.3.5.56, page 98
        } io;
    } rx;
} sio4_fasync_t;
```

4.3.5.1. sio4_fasync_t.bitrate

This field reports the data transfer bitrate. This is a read-only feature, which applied to both the transmitter and the receiver.

Valid Values

Usually, the value returned is the value passed to the `sio4_fasync_init_data()` function in the `sio4_fasync_t.bitrate` field. However, as this is 1/8th the frequency of the programmable oscillator, the reported bitrate will reflect that frequency if the oscillator is reprogrammed to an alternate frequency (section 4.3.5.5, page 47).

Low Level Function

```
int sio4_fasync_t_bitrate(
        int                     fd,
        s32*                    arg,
        sio4_fasync_action_t    action,      section 4.3.2, page 39
        FILE*                   file,
        sio4_fasync_error_t*    error);      section 4.3.3, page 39
```

The function argument and return values are described in section 4.1, page 34.

Calling Restrictions

There are no restrictions on calling this function.

4.3.5.2. sio4_fasync_t.board_leds

This field controls which of the board level LEDs are illuminated and which are not. These are board level resources common to all four channels. If settings are applied by an application for one channel the setting can be overwritten by another application accessing any of the board's other three channels.

Valid Values

Valid values are those within the inclusive range of 0x0 through 0x7. If a bit is set, then the corresponding LED is illuminated. If a bit is clear, then the LED is not illuminated. If a service is being used which applies a setting, then the value `-1` can be used to retrieve the current setting without applying or changing the current setting.

Low Level Function

```
int sio4_fasync_t_board_leds(
        int                     fd,
        s32*                    arg,
        sio4_fasync_action_t    action,       section 4.3.2, page 39
        FILE*                   file,
        sio4_fasync_error_t*    error);       section 4.3.3, page 39
```

The function argument and return values are described in section 4.1, page 34.

Calling Restrictions

There are no restrictions on calling this function.

4.3.5.3. sio4_fasync_t.chan_leds

This field configures the channel specific LEDs. Each channel has two LEDs whose states are configurable via this field. Refer to the board user manual for additional information.

Valid Values

Valid options are those from the table below. If a service is being used which applies a setting, then the value `-1` can be used to retrieve the current setting without applying or changing the current setting.

Value	**Description**
`SIO4_FASYNC_CHAN_LEDS_DEFAULT`	This is the default value.
`SIO4_FASYNC_CHAN_LEDS_BOTH_OFF`	Neither LED is illuminated.
`SIO4_FASYNC_CHAN_LEDS_LOW_ON`	The lower LED is illuminated while the other is not.
`SIO4_FASYNC_CHAN_LEDS_UP_ON`	The upper LED is illuminated while the other is not.
`SIO4_FASYNC_CHAN_LEDS_BOTH_ON`	Both LEDs are illuminated.

Low Level Function

```
int sio4_fasync_t_chan_leds(
        int                     fd,
        s32*                    arg,
        sio4_fasync_action_t    action,      section 4.3.2, page 39
        FILE*                   file,
        sio4_fasync_error_t*    error);      section 4.3.3, page 39
```

The function argument and return values are described in section 4.1, page 34.

Calling Restrictions

There are no restrictions on calling this function.

4.3.5.4. sio4_fasync_t.fw_type

This field reports the firmware type for the attached serial channel. This is a read-only feature, which applied to both the transmitter and the receiver. Any application specified values are ignored. The value returned is always the firmware type for the attached serial channel.

Valid Values

Valid options returned are those from the table below.

Value	**Description**
`SIO4_FASYNC_FW_TYPE_FASYNC`	This is a Fast Async based SIO4.
`SIO4_FASYNC_FW_TYPE_SYNC`	This is a -SYNC based SIO4.
`SIO4_FASYNC_FW_TYPE_Z16C30`	This is an SIO4 with firmware for the Z16C30 dual USC chip.

Low Level Function

```
int sio4_fasync_t_fw_type(
        int                     fd,
        s32*                    arg,
        sio4_fasync_action_t    action,       section 4.3.2, page 39
        FILE*                   file,
        sio4_fasync_error_t*    error);       section 4.3.3, page 39
```

The function argument and return values are described in section 4.1, page 34.

Calling Restrictions

There are no restrictions on calling this function.

4.3.5.5. sio4_fasync_t.osc_program

This field indicates the frequency to which the programmable oscillator will be programmed. This value will initially be eight times the bitrate passed to the `sio4_fasync_init_data()` function (section 4.1.6, page 16). If this value is modified by the application, then it will change the bitrate for both the transmitter and the receiver.

Valid Values

The default value is eight times the bitrate passed to the `sio4_fasync_init_data()` function (section 4.1.6, page 16). The valid range is from eight to 80,000,000. The programmable oscillator may not produce the exact frequency requested, but it should be very close. If a service is being used which applies a setting, then the value `-1` can be used to retrieve the current setting without applying or changing the current setting.

> **NOTE:** The library only tracks the programmed oscillator frequency through the use of this field or low-level service. If the oscillator frequency is changed by any other means, then the frequency reported be the Library will be incorrect.

Low Level Function

```
int sio4_fasync_t_osc_program(
        int                      fd,
        s32*                     arg,
        sio4_fasync_action_t     action,       section 4.3.2, page 39
        FILE*                    file,
        sio4_fasync_error_t*     error);       section 4.3.3, page 39
```

The function argument and return values are described in section 4.1, page 34.

Calling Restrictions

Making a feature selection, even if it is not being changed, must be done only while the transmitter and receiver are disabled. There are otherwise no restrictions on calling this function.

4.3.5.6. sio4_fasync_t.stop_bits

This field specifies the required number of stop-bits at the end of each byte. The setting applies both to the transmitter and the receiver.

Valid Values

Valid options are those from the table below. If a service is being used which applies a setting, then the value `-1` can be used to retrieve the current setting without applying or changing the current setting.

Value	**Description**
`SIO4_FASYNC_STOP_BITS_1`	This refers to one stop-bit.
`SIO4_FASYNC_STOP_BITS_2`	This refers to two stop-bits.

Low Level Function

```
int sio4_fasync_t_stop_bits(
        int                     fd,
        s32*                    arg,
        sio4_fasync_action_t    action,        section 4.3.2, page 39
        FILE*                   file,
        sio4_fasync_error_t*    error);        section 4.3.3, page 39
```

The function argument and return values are described in section 4.1, page 34.

Calling Restrictions

Making a feature selection, even if it is not being changed, must be done only while the transmitter and receiver are disabled. There are otherwise no restrictions on calling this function.

4.3.5.7. sio4_fasync_t.cable.enable

This field enables or disables the cable transceivers.

Valid Values

Valid options are those from the table below. If a service is being used which applies a setting, then the value `-1` can be used to retrieve the current setting without applying or changing the current setting.

Value	**Description**
`SIO4_FASYNC_CABLE_ENABLE_NO`	This option disables the transceivers.
`SIO4_FASYNC_CABLE_ENABLE_YES`	This option enables the transceivers.

Low Level Function

```
int sio4_fasync_t_cable_enable(
        int                     fd,
        s32*                    arg,
        sio4_fasync_action_t    action,       section 4.3.2, page 39
        FILE*                   file,
        sio4_fasync_error_t*    error);       section 4.3.3, page 39
```

The function argument and return values are described in section 4.1, page 34.

Calling Restrictions

Making a feature selection, even if it is not being changed, must be done only while the transmitter and receiver are disabled. There are otherwise no restrictions on calling this function.

4.3.5.8. sio4_fasync_t.cable.gpio_a

This field configures the operation of the GPIO A cable interface signal.

Valid Values

Valid options are those from the table below. If a service is being used which applies a setting, then the value `-1` can be used to retrieve the current setting without applying or changing the current setting.

Value	Description
`SIO4_FASYNC_CABLE_GPIO_DISABLE`	This option disables the signal.
`SIO4_FASYNC_CABLE_GPIO_IN`	This option configures the signal as an input.
`SIO4_FASYNC_CABLE_GPIO_OUT_0`	This option configures the signal as an output driven low (0).
`SIO4_FASYNC_CABLE_GPIO_OUT_1`	This option configures the signal as an output driven high (1).

Low Level Function

```
int sio4_fasync_t_cable_gpio_a(
        int                     fd,
        s32*                    arg,
        sio4_fasync_action_t    action,      section 4.3.2, page 39
        FILE*                   file,
        sio4_fasync_error_t*    error);      section 4.3.3, page 39
```

The function argument and return values are described in section 4.1, page 34.

Calling Restrictions

There are no restrictions on calling this function.

4.3.5.9. sio4_fasync_t.cable.gpio_b

This field configures the operation of the GPIO B cable interface signal.

Valid Values

Valid options are those from the table below. If a service is being used which applies a setting, then the value `-1` can be used to retrieve the current setting without applying or changing the current setting.

Value	**Description**
`SIO4_FASYNC_CABLE_GPIO_DISABLE`	This option disables the signal.
`SIO4_FASYNC_CABLE_GPIO_IN`	This option configures the signal as an input.
`SIO4_FASYNC_CABLE_GPIO_OUT_0`	This option configures the signal as an output driven low (0).
`SIO4_FASYNC_CABLE_GPIO_OUT_1`	This option configures the signal as an output driven high (1).

Low Level Function

```
int sio4_fasync_t_cable_gpio_b(
        int                     fd,
        s32*                    arg,
        sio4_fasync_action_t    action,      section 4.3.2, page 39
        FILE*                   file,
        sio4_fasync_error_t*    error);      section 4.3.3, page 39
```

The function argument and return values are described in section 4.1, page 34.

Calling Restrictions

There are no restrictions on calling this function.

4.3.5.10. sio4_fasync_t.cable.gpio_c

This field configures the operation of the GPIO C cable interface signal.

Valid Values

Valid options are those from the table below. If a service is being used which applies a setting, then the value `-1` can be used to retrieve the current setting without applying or changing the current setting.

Value	**Description**
`SIO4_FASYNC_CABLE_GPIO_DISABLE`	This option disables the signal.
`SIO4_FASYNC_CABLE_GPIO_IN`	This option configures the signal as an input.
`SIO4_FASYNC_CABLE_GPIO_OUT_0`	This option configures the signal as an output driven low (0).
`SIO4_FASYNC_CABLE_GPIO_OUT_1`	This option configures the signal as an output driven high (1).

Low Level Function

```
int sio4_fasync_t_cable_gpio_c(
        int                     fd,
        s32*                    arg,
        sio4_fasync_action_t    action,      section 4.3.2, page 39
        FILE*                   file,
        sio4_fasync_error_t*    error);      section 4.3.3, page 39
```

The function argument and return values are described in section 4.1, page 34.

Calling Restrictions

There are no restrictions on calling this function.

4.3.5.11. sio4_fasync_t.cable.gpio_d

This field configures the operation of the GPIO D cable interface signal.

Valid Values

Valid options are those from the table below. If a service is being used which applies a setting, then the value `-1` can be used to retrieve the current setting without applying or changing the current setting.

Value	**Description**
`SIO4_FASYNC_CABLE_GPIO_DISABLE`	This option disables the signal.
`SIO4_FASYNC_CABLE_GPIO_IN`	This option configures the signal as an input.
`SIO4_FASYNC_CABLE_GPIO_OUT_0`	This option configures the signal as an output driven low (0).
`SIO4_FASYNC_CABLE_GPIO_OUT_1`	This option configures the signal as an output driven high (1).

Low Level Function

```
int sio4_fasync_t_cable_gpio_d(
        int                     fd,
        s32*                    arg,
        sio4_fasync_action_t    action,      section 4.3.2, page 39
        FILE*                   file,
        sio4_fasync_error_t*    error);      section 4.3.3, page 39
```

The function argument and return values are described in section 4.1, page 34.

Calling Restrictions

There are no restrictions on calling this function.

4.3.5.12. sio4_fasync_t.cable.loopback

This field configures the cable loopback feature, which affects the Tx and Rx Data signals, as well as the CTS and RTS hardware flow control signals.

Valid Values

Valid options are those from the table below. If a service is being used which applies a setting, then the value `-1` can be used to retrieve the current setting without applying or changing the current setting.

Value	**Description**
`SIO4_FASYNC_CABLE_LOOPBACK_NO`	This option disables the loopback feature.
`SIO4_FASYNC_CABLE_LOOPBACK_INT`	This option enables internal loopback operation, in which the signals are looped back within the firmware. †
`SIO4_FASYNC_CABLE_LOOPBACK_EXT`	This option enables external loopback operation, in which the input signals are redirected from the input transceivers to the input side of the corresponding output transceivers. This option requires that the cable transceivers be enabled (section 4.3.5.7, page 48).

† If the transceivers are enabled, then the TxD and RTS cable signals are driven accordingly.

Low Level Function

```
int sio4_fasync_t_cable_loopback(
        int                     fd,
        s32*                    arg,
        sio4_fasync_action_t    action,      section 4.3.2, page 39
        FILE*                   file,
        sio4_fasync_error_t*    error);      section 4.3.3, page 39
```

The function argument and return values are described in section 4.1, page 34.

Calling Restrictions

Making a feature selection, even if it is not being changed, must be done only while the transceivers are disabled. There are otherwise no restrictions on calling this function.

4.3.5.13. sio4_fasync_t.cable.mode

This field selects the between the DCE and DTE cable signal configurations. Refer to the board user manual for clarification. This affects the cable signals for both the transmitter and the receiver.

Valid Values

Valid options are those from the table below. If a service is being used which applies a setting, then the value `-1` can be used to retrieve the current setting without applying or changing the current setting.

Value	**Description**
`SIO4_FASYNC_CABLE_MODE_DCE`	This option selects the DCE cable signal configuration.
`SIO4_FASYNC_CABLE_MODE_DTE`	This option selects the DTE cable signal configuration.

Low Level Function

```
int sio4_fasync_t_cable_mode(
        int                     fd,
        s32*                    arg,
        sio4_fasync_action_t    action,      section 4.3.2, page 39
        FILE*                   file,
        sio4_fasync_error_t*    error);      section 4.3.3, page 39
```

The function argument and return values are described in section 4.1, page 34.

Calling Restrictions

Making a feature selection, even if it is not being changed, must be done only while the transmitter and receiver are disabled. There are otherwise no restrictions on calling this function.

4.3.5.14. sio4_fasync_t.cable.protocol

This field selects the transceiver signal protocol used at the cable interface once the cable transceivers are enabled. This is a read-only feature.

Valid Values

Valid options are those from the table below. If a service is being used which applies a setting, then the value `-1` can be used to retrieve the current setting without applying or changing the current setting.

Value	**Description**
`SIO4_FASYNC_CABLE_PROTOCOL_RS422_RS485`	This selects the RS-422/RS-485 cable signal protocol. This is the default, and at this time, it is the only supported option.

Low Level Function

```
int sio4_fasync_t_cable_protocol(
        int                     fd,
        s32*                    arg,
        sio4_fasync_action_t    action,       section 4.3.2, page 39
        FILE*                   file,
        sio4_fasync_error_t*    error);       section 4.3.3, page 39
```

The function argument and return values are described in section 4.1, page 34.

Calling Restrictions

There are no restrictions on calling this function.

4.3.5.15. sio4_fasync_t.cable.term

This field enables or disables the termination resisters used at the cable interface once the cable transceivers are enabled. Refer to the board user manual for clarification.

> **NOTE:** This feature refers to the termination resistors internal to the multi-protocol transceiver chips used on the SIO4. These are totally separate from the socketed external termination resistor packs which may also be present on the board.

Valid Values

Valid options are those from the table below.

Value	**Description**
`SIO4_FASYNC_CABLE_TERM_DISABLE`	This disables the internal termination resisters.
`SIO4_FASYNC_CABLE_TERM_ENABLE`	This enables the internal termination resisters.

Low Level Function

```
int sio4_fasync_t_cable_term(
        int                      fd,
        s32*                     arg,
        sio4_fasync_action_t     action,       section 4.3.2, page 39
        FILE*                    file,
        sio4_fasync_error_t*     error);       section 4.3.3, page 39
```

The function argument and return values are described in section 4.1, page 34.

Calling Restrictions

There are no restrictions on calling this function.

4.3.5.16. sio4_fasync_t.irq.direction

This field configures interrupts as rising or falling edge sensitive. If a bit is set then the respective interrupt is triggered by the condition becoming asserted (a rising edge). If a bit is clear then the respective interrupt is triggered by the condition becoming negated (a falling edge).

Valid Values

Valid values are any combination of the Interrupt Enable options given in section 4.3.5.17, page 59. If a service is being used which applies a setting, then the value `-1` can be used to retrieve the current setting without applying or changing the current setting.

Low Level Function

```
int sio4_fasync_t_irq_direction(
        int                     fd,
        s32*                    arg,
        sio4_fasync_action_t    action,      section 4.3.2, page 39
        FILE*                   file,
        sio4_fasync_error_t*    error);      section 4.3.3, page 39
```

The function argument and return values are described in section 4.1, page 34.

Calling Restrictions

There are no restrictions on calling this function.

4.3.5.17. sio4_fasync_t.irq.enable

This field enables or disables channel interrupts. If a bit is set, then that interrupt is enabled. The interrupt is otherwise disabled.

Valid Values

Valid values are any combination of the macros in the following table. If a service is being used which applies a setting, then the value `-1` can be used to retrieve the current setting without applying or changing the current setting.

Value	**Description**
`SIO4_FASYNC_IRQ_ALL`	This refers to all of the interrupt in this table.
`SIO4_FASYNC_IRQ_RX_FIFO_AF`	This refers to the Rx FIFO Almost Full interrupt.
`SIO4_FASYNC_IRQ_RX_FIFO_EMPTY`	This refers to the Rx FIFO Empty interrupt.
`SIO4_FASYNC_IRQ_RX_FIFO_FULL`	This refers to the Rx FIFO Full interrupt.
`SIO4_FASYNC_IRQ_RX_FRAME_ERROR`	This refers to the Rx Frame Error interrupt.
`SIO4_FASYNC_IRQ_RX_PARITY_ERROR`	This refers to the Rx Parity Error interrupt.
`SIO4_FASYNC_IRQ_TX_FIFO_AE`	This refers to the Tx FIFO Almost Empty interrupt.
`SIO4_FASYNC_IRQ_TX_FIFO_EMPTY`	This refers to the Tx FIFO Empty interrupt.
`SIO4_FASYNC_IRQ_TX_FIFO_FULL`	This refers to the Tx FIFO Full interrupt.

Low Level Function

```
int sio4_fasync_t_irq_enable(
        int                     fd,
        s32*                    arg,
        sio4_fasync_action_t    action,       section 4.3.2, page 39
        FILE*                   file,
        sio4_fasync_error_t*    error);       section 4.3.3, page 39
```

The function argument and return values are described in section 4.1, page 34.

Calling Restrictions

There are no restrictions on calling this function.

4.3.5.18. sio4_fasync_t.parity.enable

This field enables or disables the use of parity. This setting applies both to the transmitter and the receiver.

Valid Values

Valid options are those from the table below. If a service is being used which applies a setting, then the value `-1` can be used to retrieve the current setting without applying or changing the current setting.

Value	**Description**
`SIO4_FASYNC_PARITY_ENABLE_NO`	This option disables the use of parity.
`SIO4_FASYNC_PARITY_ENABLE_YES`	This option enables the use of parity.

Low Level Function

```
int sio4_fasync_t_parity_enable(
        int                     fd,
        s32*                    arg,
        sio4_fasync_action_t    action,      section 4.3.2, page 39
        FILE*                   file,
        sio4_fasync_error_t*    error);      section 4.3.3, page 39
```

The function argument and return values are described in section 4.1, page 34.

Calling Restrictions

Making a feature selection, even if it is not being changed, must be done only while the transmitter and receiver are disabled. There are otherwise no restrictions on calling this function.

4.3.5.19. sio4_fasync_t.parity.type

This field configures the type of parity used, when enabled. This setting applies both to the transmitter and the receiver.

Valid Values

Valid options are those from the table below. If a service is being used which applies a setting, then the value `-1` can be used to retrieve the current setting without applying or changing the current setting.

Value	**Description**
`SIO4_FASYNC_PARITY_TYPE_EVEN`	This option refers to Even parity.
`SIO4_FASYNC_PARITY_TYPE_MARK`	This option refers to Mark parity.
`SIO4_FASYNC_PARITY_TYPE_ODD`	This option refers to Odd parity.
`SIO4_FASYNC_PARITY_TYPE_SPACE`	This option refers to Space parity.

Low Level Function

```
int sio4_fasync_t_parity_type(
        int                     fd,
        s32*                    arg,
        sio4_fasync_action_t    action,       section 4.3.2, page 39
        FILE*                   file,
        sio4_fasync_error_t*    error);       section 4.3.3, page 39
```

The function argument and return values are described in section 4.1, page 34.

Calling Restrictions

Making a feature selection, even if it is not being changed, must be done only while the transmitter and receiver are disabled. There are otherwise no restrictions on calling this function.

4.3.5.20. sio4_fasync_t.tx.cts

This field configures the operation of the Tx CTS cable interface signal.

Valid Values

Valid options are those from the table below. If a service is being used which applies a setting, then the value `-1` can be used to retrieve the current setting without applying or changing the current setting.

Value	**Description**
`SIO4_FASYNC_TX_CTS_IN`	The signal is an input which does not affect data transmission.
`SIO4_FASYNC_TX_CTS_IN_AUTO_TX`	The signal is an input which, when asserted, alerts the transmitter to send data.

Low Level Function

```
int sio4_fasync_t_tx_cts(
        int                     fd,
        s32*                    arg,
        sio4_fasync_action_t    action,      section 4.3.2, page 39
        FILE*                   file,
        sio4_fasync_error_t*    error);      section 4.3.3, page 39
```

The function argument and return values are described in section 4.1, page 34.

Calling Restrictions

Making a feature selection, even if it is not being changed, must be done only while the transmitter is disabled. There are otherwise no restrictions on calling this function.

4.3.5.21. sio4_fasync_t.tx.data

This field configures the operation of the Tx Data cable interface signal.

Valid Values

Valid options are those from the table below. If a service is being used which applies a setting, then the value `-1` can be used to retrieve the current setting without applying or changing the current setting.

> **NOTE:** The Tx Data signal is driven only when the cable transceivers are enabled (section 4.3.5.7, page 49) and are driven as configured here only when the transmitter is also enabled (section 4.3.5.22, page 64).

Value	**Description**
`SIO4_FASYNC_TX_DATA_OUT_0`	This drives the signal low.
`SIO4_FASYNC_TX_DATA_OUT_1`	This drives the signal high.
`SIO4_FASYNC_TX_DATA_OUT_AUTO`	This drives the signal as necessary to trans data.

Low Level Function

```
int sio4_fasync_t_tx_data(
        int                     fd,
        s32*                    arg,
        sio4_fasync_action_t    action,      section 4.3.2, page 39
        FILE*                   file,
        sio4_fasync_error_t*    error);      section 4.3.3, page 39
```

The function argument and return values are described in section 4.1, page 34.

Calling Restrictions

There are no restrictions on calling this function.

4.3.5.22. sio4_fasync_t.tx.enable

This field enables and disables the transmitter. When requesting that the transmitter be enabled, it becomes enabled immediately. When requesting that it be disabled, it becomes disabled immediately only if it is idle. If it is busy sending a byte, then it becomes disabled when transmission of that byte is complete, including any added Stop Bits (section 4.3.5.23, page 65). When disabling the transmitter, the Library will wait for up to one second for the transmitter to become disabled. If it is not disabled within that time frame, then an error will be reported.

> **NOTE:** The transmitter is busy if it is actively engaged in sending out a data byte or any part of it. This includes the Start Bit, the data bits, the Parity Bit, the required Stop Bits and any extra Stop Bits. The transmitter is otherwise idle.

Valid Values

Valid options are those from the table below. If a service is being used which applies a setting, then the value `-1` can be used to retrieve the current setting without applying or changing the current setting.

Value	**Description**
`SIO4_FASYNC_TX_ENABLE_NO`	This option disables the transmitter.
`SIO4_FASYNC_TX_ENABLE_YES`	This option enables the transmitter.

Low Level Function

```
int sio4_fasync_t_tx_enable(
        int                     fd,
        s32*                    arg,
        sio4_fasync_action_t    action,      section 4.3.2, page 39
        FILE*                   file,
        sio4_fasync_error_t*    error);      section 4.3.3, page 39
```

The function argument and return values are described in section 4.1, page 34.

Calling Restrictions

Making a feature selection, even if it is not being changed, must be done only while the cable transceivers are enabled. There are otherwise no restrictions on calling this function.

4.3.5.23. sio4_fasync_t.tx.gap

This field specifies the number of additional stop-bit periods the transmitter inserts after sending the required stop bits (section 4.3.5.6, page 48) for each byte.

Valid Values

Valid values are those within the inclusive range of zero to 0xFFFF. If a service is being used which applies a setting, then the value `-1` can be used to retrieve the current setting without applying or changing the current setting.

Low Level Function

```
int sio4_fasync_t_tx_gap(
        int                     fd,
        s32*                    arg,
        sio4_fasync_action_t    action,      section 4.3.2, page 39
        FILE*                   file,
        sio4_fasync_error_t*    error);      section 4.3.3, page 39
```

The function argument and return values are described in section 4.1, page 34.

Calling Restrictions

Making a feature selection, even if it is not being changed, must be done only while the transmitter is disabled. The exception to this is applying the value zero. The value zero can be applied at any time and will take effect immediately. In addition, it will also abort the operation if it the transmitter is actively working on the configured Gap period. There are otherwise no restrictions on calling this function.

4.3.5.24. sio4_fasync_t.tx.status

This field reports the state of the transmitter. This is a read-only feature.

Valid Values

Valid options returned are those from the table below.

The transmitter is busy if it is actively engaged in sending out a data byte or any part of it. This includes the Start Bit, the data bits, the Parity Bit, the required Stop Bits and any extra Stop Bits. The transmitter is otherwise idle.

Value	**Description**
`SIO4_FASYNC_TX_STATUS_DISABLED`	The transmitter is disabled.
`SIO4_FASYNC_TX_STATUS_IDLE`	The transmitter is enabled, but not in the process of sending a byte.
`SIO4_FASYNC_TX_STATUS_BUSY`	The transmitter is enabled and in the process of sending a byte. †

† The transmitter is busy if it is actively engaged in sending out a data byte or any part of it. This includes the Start Bit, the data bits, the Parity Bit, the required Stop Bits and any extra Stop Bits.

Low Level Function

```
int sio4_fasync_t_tx_status(
        int                     fd,
        s32*                    arg,
        sio4_fasync_action_t    action,      section 4.3.2, page 39
        FILE*                   file,
        sio4_fasync_error_t*    error);      section 4.3.3, page 39
```

The function argument and return values are described in section 4.1, page 34.

Calling Restrictions

There are no restrictions on calling this function.

4.3.5.25. sio4_fasync_t.tx.fifo.ae

This field configures the Tx FIFO Almost Empty threshold level. When applying a setting the FIFO is reset and the current content is lost. The Tx FIFO Almost Empty status is asserted when the Tx FIFO contains *Almost Empty* or fewer data values.

Valid Values

Valid values are those within the inclusive range of zero to 0xFFFF. If a service is being used which applies a setting, then the value `-1` can be used to retrieve the current setting without applying or changing the current setting.

Low Level Function

```
int sio4_fasync_t_tx_fifo_ae(
        int                     fd,
        s32*                    arg,
        sio4_fasync_action_t    action,       section 4.3.2, page 39
        FILE*                   file,
        sio4_fasync_error_t*    error);       section 4.3.3, page 39
```

The function argument and return values are described in section 4.1, page 34.

Calling Restrictions

There are no restrictions on calling this function.

4.3.5.26. sio4_fasync_t.tx.fifo.af

This field configures the Tx FIFO Almost Full threshold level. When applying a setting the Tx FIFO is reset and the current content is lost. The Tx FIFO Almost Full status is asserted when the Tx FIFO can accept *Almost Full* or fewer data values before becoming full.

Valid Values

Valid values are those within the inclusive range of zero to 0xFFFF. If a service is being used which applies a setting, then the value `-1` can be used to retrieve the current setting without applying or changing the current setting.

Low Level Function

```
int sio4_fasync_t_tx_fifo_af(
        int                     fd,
        s32*                    arg,
        sio4_fasync_action_t    action,       section 4.3.2, page 39
        FILE*                   file,
        sio4_fasync_error_t*    error);       section 4.3.3, page 39
```

The function argument and return values are described in section 4.1, page 34.

Calling Restrictions

There are no restrictions on calling this function.

4.3.5.27. sio4_fasync_t.tx.fifo.bytes

This field reports the number of bytes in the Tx FIFO. This is a read-only feature.

Valid Values

Valid values returned are those within the inclusive range of zero up through the size of the Tx FIFO.

Low Level Function

```
int sio4_fasync_t_tx_fifo_bytes(
        int                     fd,
        s32*                    arg,
        sio4_fasync_action_t    action,       section 4.3.2, page 39
        FILE*                   file,
        sio4_fasync_error_t*    error);       section 4.3.3, page 39
```

The function argument and return values are described in section 4.1, page 34.

Calling Restrictions

There are no restrictions on calling this function.

4.3.5.28. sio4_fasync_t.tx.fifo.empty_cfg

This field configures the transmitter's response to the Tx FIFO becoming empty.

Valid Values

Valid options are those from the table below. If a service is being used which applies a setting, then the value `-1` can be used to retrieve the current setting without applying or changing the current setting.

Value	**Description**
`SIO4_FASYNC_TX_FIFO_EMPTY_CFG_DISABLE`	With this option, the transmitter will be disabled. †
`SIO4_FASYNC_TX_FIFO_EMPTY_CFG_ENABLE`	With this option, the transmitter will remain enabled.

† Using this option may call for preloading Tx data into the Tx FIFO. For details see section 5.5, page 102.

Low Level Function

```
int sio4_fasync_t_tx_fifo_empty_cfg(
        int                     fd,
        s32*                    arg,
        sio4_fasync_action_t    action,       section 4.3.2, page 39
        FILE*                   file,
        sio4_fasync_error_t*    error);       section 4.3.3, page 39
```

The function argument and return values are described in section 4.1, page 34.

Calling Restrictions

Making a feature selection, even if it is not being changed, must be done only while the transmitter is disabled. There are otherwise no restrictions on calling this function.

4.3.5.29. sio4_fasync_t.tx.fifo.overflow

This field pertains to the Tx FIFO overflow status. The service will always return the current overflow status.

Valid Values

Valid options are those from the table below. If a service is being used which applies a setting, then the value `-1` can be used to retrieve the current setting without applying or changing the current setting.

Value	**Description**
`SIO4_FASYNC_FIFO_OVERFLOW_CLEAR`	This option is used to clear the overflow status.
`SIO4_FASYNC_FIFO_OVERFLOW_TEST`	This option is used to request the overflow status.
`SIO4_FASYNC_FIFO_OVERFLOW_NO`	This option is returned when the current status indicates that an overflow has not occurred.
`SIO4_FASYNC_FIFO_OVERFLOW_YES`	This option is returned when the current status indicates that an overflow has occurred.

Low Level Function

```
int sio4_fasync_t_tx_fifo_overflow(
        int                     fd,
        s32*                    arg,
        sio4_fasync_action_t    action,       section 4.3.2, page 39
        FILE*                   file,
        sio4_fasync_error_t*    error);       section 4.3.3, page 39
```

The function argument and return values are described in section 4.1, page 34.

Calling Restrictions

There are no restrictions on calling this function.

4.3.5.30. sio4_fasync_t.tx.fifo.size

This field reports the size the Tx FIFO in bytes. This is a read-only feature.

Valid Values

Valid values returned should be within the inclusive range from 512 to 32,768.

Low Level Function

```
int sio4_fasync_t_tx_fifo_size(
        int                     fd,
        s32*                    arg,
        sio4_fasync_action_t    action,     section 4.3.2, page 39
        FILE*                   file,
        sio4_fasync_error_t*    error);     section 4.3.3, page 39
```

The function argument and return values are described in section 4.1, page 34.

Calling Restrictions

There are no restrictions on calling this function.

4.3.5.31. sio4_fasync_t.tx.fifo.status

This field reports the relative fill level of the Tx FIFO. This is a read-only feature.

Valid Values

Valid options returned are those from the table below.

Value	**Description**
`SIO4_FASYNC_FIFO_STATUS_EMPTY`	The FIFO is empty.
`SIO4_FASYNC_FIFO_STATUS_AE`	The FIFO is at or below the Tx FIFO Almost Empty level (section 4.3.5.25, page 67), though it is not empty.
`SIO4_FASYNC_FIFO_STATUS_MEDIUM`	The FIFO fill level is between the Tx FIFO Almost Empty and Tx FIFO Almost Full levels.
`SIO4_FASYNC_FIFO_STATUS_AF`	The FIFO is at or above the Tx FIFO Almost Full level (section 4.3.5.26, page 68), though it is not full.
`SIO4_FASYNC_FIFO_STATUS_FULL`	The FIFO is full.

Low Level Function

```
int sio4_fasync_t_tx_fifo_status(
        int                     fd,
        s32*                    arg,
        sio4_fasync_action_t    action,       section 4.3.2, page 39
        FILE*                   file,
        sio4_fasync_error_t*    error);       section 4.3.3, page 39
```

The function argument and return values are described in section 4.1, page 34.

Calling Restrictions

There are no restrictions on calling this function.

4.3.5.32. sio4_fasync_t.tx.io.dma_threshold

This field specifies the minimum size of Block Mode DMA transfers performed by the driver. The device driver’s write service considers the size of the write request and the size of individual DMA requests before proceeding with a Block Mode DMA transfer. If the Tx FIFO has sufficient space to satisfy the write request, then the driver proceeds. Otherwise, if the size of the individual DMA request is below this threshold, then the driver will wait 1ms for more Tx FIFO space to become available and then try again.

> **NOTE:** The driver may break individual Block Mode DMA write requests into multiple, smaller Block Mode DMA transfers. This is based on the size of the write request and the amount of available space in the Tx FIFO at the moment the Tx FIFO state is examined.

Valid Values

Valid values are the inclusive range from zero to the size of the Tx FIFO. If a service is being used which applies a setting, then the value `-1` can be used to retrieve the current setting without applying or changing the current setting.

Low Level Function

```
int sio4_fasync_t_tx_io_dma_threshold(
        int                     fd,
        s32*                    arg,
        sio4_fasync_action_t    action,      section 4.3.2, page 39
        FILE*                   file,
        sio4_fasync_error_t*    error);      section 4.3.3, page 39
```

The function argument and return values are described in section 4.1, page 34.

Calling Restrictions

There are no restrictions on calling this function.

4.3.5.33. sio4_fasync_t.tx.io.mode

This field selects the data transfer mode to be used for write requests.

Valid Values

Valid options are those from the table below. These macros are defined in the header file `gsc_common.h`. If a service is being used which applies a setting, then the value `-1` can be used to retrieve the current setting without applying or changing the current setting.

Value	Description
`GSC_IO_MODE_BMDMA`	This option refers to Block Mode DMA. †
`GSC_IO_MODE_DMDMA`	This mode refers to Demand Mode DMA. †
`GSC_IO_MODE_PIO`	This mode refers to PIO.

† The SIO4 has only two DMA engines. A BMDMA or DMDMA transfer request may fail if both DMA engines are already in use by other SIO4 I/O requests.

Low Level Function

```
int sio4_fasync_t_tx_io_mode(
        int                     fd,
        s32*                    arg,
        sio4_fasync_action_t    action,       section 4.3.2, page 39
        FILE*                   file,
        sio4_fasync_error_t*    error);       section 4.3.3, page 39
```

The function argument and return values are described in section 4.1, page 34.

Calling Restrictions

There are no restrictions on calling this function.

4.3.5.34. sio4_fasync_t.tx.io.overflow

This field specifies if the driver's write service should account for Tx FIFO Overflow conditions. The overflow check, when enabled, is performed upon entry to the driver's write service.

Valid Values

Valid options are those from the table below. If a service is being used which applies a setting, then the value `-1` can be used to retrieve the current setting without applying or changing the current setting.

Value	**Description**
`SIO4_FASYNC_IO_OVERFLOW_CHECK`	The driver checks for Tx FIFO overflows. †
`SIO4_FASYNC_IO_OVERFLOW_IGNORE`	The driver does not check for Tx FIFO overflows.

† If an overflow is detected, then the write request immediately returns an error status.

Low Level Function

```
int sio4_fasync_t_tx_io_overflow(
        int                     fd,
        s32*                    arg,
        sio4_fasync_action_t    action,       section 4.3.2, page 39
        FILE*                   file,
        sio4_fasync_error_t*    error);       section 4.3.3, page 39
```

The function argument and return values are described in section 4.1, page 34.

Calling Restrictions

There are no restrictions on calling this function.

4.3.5.35. sio4_fasync_t.tx.io.pio_threshold

This field specifies the threshold below which individual DMA based write requests and individual DMA transfer requests will instead use PIO. This mode change is made because very small DMA requests can be performed more quickly by using PIO.

> **NOTE:** The driver redirects individual DMA write requests and individual DMA transfer requests to use PIO when the volume of such requests is less than the configured threshold.

Valid Values

Valid values are the inclusive range from zero to 0xFFFFFFF. A value of zero disables the change to PIO. A value equal to or larger than the size of the Tx FIFO will essentially force all Block Mode DMA write requests to use PIO. A value equal to or larger than the 64K (an internal driver limit) will essentially force all Demand Mode DMA write requests to use PIO. If a service is being used which applies a setting, then the value `-1` can be used to retrieve the current setting without applying or changing the current setting.

Low Level Function

```
int sio4_fasync_t_tx_io_pio_threshold(
        int                     fd,
        s32*                    arg,
        sio4_fasync_action_t    action,      section 4.3.2, page 39
        FILE*                   file,
        sio4_fasync_error_t*    error);      section 4.3.3, page 39
```

The function argument and return values are described in section 4.1, page 34.

Calling Restrictions

There are no restrictions on calling this function.

4.3.5.36. sio4_fasync_t.tx.io.timeout

This field specifies the maximum amount of time, in seconds, that the driver permits for write requests. The driver's write service will return either when the request has been satisfied or when the timeout period expires, whichever occurs first.

Valid Values

Valid options are those in the range from the minimum to the maximum, as shown below, plus the *infinite* option. These macros are defined in header file `sio4.h`. If a service is being used which applies a setting, then the value `-1` can be used to retrieve the current setting without applying or changing the current setting.

Value	**Description**
`SIO4_IO_TIMEOUT_DEFAULT`	This is the default, which is 10 seconds.
`SIO4_IO_TIMEOUT_INFINITE`	This tells the driver to never timeout.
`SIO4_IO_TIMEOUT_MAX`	This is the maximum amount of time permitted, which is 3,600 seconds, or one hour.
`SIO4_IO_TIMEOUT_MIN`	This is the minimum, which is zero, and is the same as the *No Sleep* option.
`SIO4_IO_TIMEOUT_NO_SLEEP`	This tells the driver not to sleep to wait for more space in the Tx FIFO. In this case, the service returns rather than wait for more space.

Low Level Function

```
int sio4_fasync_t_tx_io_timeout(
        int                     fd,
        s32*                    arg,
        sio4_fasync_action_t    action,     section 4.3.2, page 39
        FILE*                   file,
        sio4_fasync_error_t*    error);     section 4.3.3, page 39
```

The function argument and return values are described in section 4.1, page 34.

Calling Restrictions

There are no restrictions on calling this function.

4.3.5.37. sio4_fasync_t.rx.enable

This field enables and disables the receiver. When requesting that the receiver be enabled, it becomes enabled immediately. When requesting that it be disabled, it becomes disabled immediately only if it is idle. If it is busy receiving a byte, then it becomes disabled when reception of that byte is complete, up through and including the required Stop Bits. When disabling the receiver, the Library will wait for up to one second for the receiver to become disabled. If it is not disabled within that time frame, then an error will be reported.

> **NOTE:** The receiver is considered idle during the first half of the Start Bit, which is the point at which the Start Bit has been verified as not being noise.

Valid Values

Valid options are those from the table below. If a service is being used which applies a setting, then the value `-1` can be used to retrieve the current setting without applying or changing the current setting.

Value	**Description**
`SIO4_FASYNC_RX_ENABLE_NO`	This option disables the receiver.
`SIO4_FASYNC_RX_ENABLE_YES`	This option enables the receiver.

Low Level Function

```
int sio4_fasync_t_rx_enable(
        int                     fd,
        s32*                    arg,
        sio4_fasync_action_t    action,       section 4.3.2, page 39
        FILE*                   file,
        sio4_fasync_error_t*    error);       section 4.3.3, page 39
```

The function argument and return values are described in section 4.1, page 34.

Calling Restrictions

Making a feature selection, even if it is not being changed, must be done only while the cable transceivers are enabled and the cable mode (DCE/DTE) has been set . There are otherwise no restrictions on calling this function.

4.3.5.38. sio4_fasync_t.rx.frame_err_cfg

This field configures the receiver’s response to Rx Frame errors.

Valid Values

Valid options are those from the table below. If a service is being used which applies a setting, then the value `-1` can be used to retrieve the current setting without applying or changing the current setting.

Value	**Description**
`SIO4_FASYNC_RX_ERROR_CFG_DISCARD`	With this option, the byte being received is discarded rather than being transferred to the Rx FIFO.
`SIO4_FASYNC_RX_ERROR_CFG_SAVE`	With this option, the byte being received is transferred to the Rx FIFO.

Low Level Function

```
int sio4_fasync_t_rx_frame_err_cfg(
        int                     fd,
        s32*                    arg,
        sio4_fasync_action_t    action,      section 4.3.2, page 39
        FILE*                   file,
        sio4_fasync_error_t*    error);      section 4.3.3, page 39
```

The function argument and return values are described in section 4.1, page 34.

Calling Restrictions

Making a feature selection, even if it is not being changed, must be done only while the receiver is disabled. There are otherwise no restrictions on calling this function.

4.3.5.39. sio4_fasync_t.rx.parity_err_cfg

This field configures the receiver’s response to Rx Parity errors.

Valid Values

Valid options are those from the table below. If a service is being used which applies a setting, then the value `-1` can be used to retrieve the current setting without applying or changing the current setting.

Value	**Description**
`SIO4_FASYNC_RX_ERROR_CFG_DISCARD`	With this option, the byte being received is discarded rather than being transferred to the Rx FIFO.
`SIO4_FASYNC_RX_ERROR_CFG_SAVE`	With this option, the byte being received is transferred to the Rx FIFO.

Low Level Function

```
int sio4_fasync_t_rx_parity_err_cfg(
        int                     fd,
        s32*                    arg,
        sio4_fasync_action_t    action,       section 4.3.2, page 39
        FILE*                   file,
        sio4_fasync_error_t*    error);       section 4.3.3, page 39
```

The function argument and return values are described in section 4.1, page 34.

Calling Restrictions

Making a feature selection, even if it is not being changed, must be done only while the receiver is disabled. There are otherwise no restrictions on calling this function.

4.3.5.40. sio4_fasync_t.rx.rts

This field configures the operation of the Rx RTS cable interface signal.

Valid Values

Valid options are those from the table below. If a service is being used which applies a setting, then the value `-1` can be used to retrieve the current setting without applying or changing the current setting.

> **NOTE:** The Rx RTS signal is driven only when the cable transceivers are enabled (section 4.3.5.7, page 49) and are driven as configured here only when the transmitter is also enabled (section 4.3.5.22, page 64).

Value	**Description**
`SIO4_FASYNC_RX_RTS_DISABLE`	The RTS signal is disabled and the cable interface signal is driven in a negated state.
`SIO4_FASYNC_RX_RTS_OUT_0`	This drives the signal low.
`SIO4_FASYNC_RX_RTS_OUT_1`	This drives the signal high.
`SIO4_FASYNC_RX_RTS_OUT_RX_ENABLE`	The signal is driven hi when the receiver is enabled and is driven low when the receiver is disabled.
`SIO4_FASYNC_RX_RTS_OUT_RX_F_NAF`	The signal is driven high if the Rx FIFO is not Almost Full and is driven low when the Rx FIFO is Almost Full.

Low Level Function

```
int sio4_fasync_t_rx_rts(
        int                     fd,
        s32*                    arg,
        sio4_fasync_action_t    action,      section 4.3.2, page 39
        FILE*                   file,
        sio4_fasync_error_t*    error);      section 4.3.3, page 39
```

The function argument and return values are described in section 4.1, page 34.

Calling Restrictions

There are no restrictions on calling this function.

4.3.5.41. sio4_fasync_t.rx.sb2_err_cfg

This field specifies how the receiver should respond if two stop bits are expected, but only one is received.

Valid Values

Valid options are those from the table below. If a service is being used which applies a setting, then the value `-1` can be used to retrieve the current setting without applying or changing the current setting.

Value	Description
`SIO4_FASYNC_RX_SB2_ERR_CFG_ERR`	The occurrence should generate a frame error.
`SIO4_FASYNC_RX_SB2_ERR_CFG_VAL`	The occurrence is valid and should be ignored.

Low Level Function

```
int sio4_fasync_t_rx_sb2_err_cfg(
        int                     fd,
        s32*                    arg,
        sio4_fasync_action_t    action,       section 4.3.2, page 39
        FILE*                   file,
        sio4_fasync_error_t*    error);       section 4.3.3, page 39
```

The function argument and return values are described in section 4.1, page 34.

Calling Restrictions

Making a feature selection, even if it is not being changed, must be done only while the receiver is disabled. There are otherwise no restrictions on calling this function.

4.3.5.42. sio4_fasync_t.rx.status

This field reports the state of the receiver. This is a read-only feature.

Valid Values

Valid options returned are those from the table below.

Value	**Description**
`SIO4_FASYNC_RX_STATUS_DISABLED`	The receiver is disabled.
`SIO4_FASYNC_RX_STATUS_IDLE`	The receiver is enabled, but not in the process of receiving a byte.
`SIO4_FASYNC_RX_STATUS_BUSY`	The receiver is enabled and in the process of receiving a byte. †

† The receiver is busy if it is actively engaged in taking in a data byte or any part of it. This includes the Start Bit, the data bits, the Parity Bit and the required Stop Bits.

Low Level Function

```
int sio4_fasync_t_rx_status(
        int                     fd,
        s32*                    arg,
        sio4_fasync_action_t    action,       section 4.3.2, page 39
        FILE*                   file,
        sio4_fasync_error_t*    error);       section 4.3.3, page 39
```

The function argument and return values are described in section 4.1, page 34.

Calling Restrictions

There are no restrictions on calling this function.

4.3.5.43. sio4_fasync_t.rx.fifo.ae

This field configures the Rx FIFO Almost Empty threshold level. When applying a setting the FIFO is reset and the current content is lost. The Rx FIFO Almost Empty status is asserted when the Rx FIFO contains *Almost Empty* or fewer data values.

Valid Values

Valid values are those within the inclusive range of zero to 0xFFFF. If a service is being used which applies a setting, then the value `-1` can be used to retrieve the current setting without applying or changing the current setting.

Low Level Function

```
int sio4_fasync_t_rx_fifo_ae(
        int                     fd,
        s32*                    arg,
        sio4_fasync_action_t    action,       section 4.3.2, page 39
        FILE*                   file,
        sio4_fasync_error_t*    error);       section 4.3.3, page 39
```

The function argument and return values are described in section 4.1, page 34.

Calling Restrictions

There are no restrictions on calling this function.

4.3.5.44. sio4_fasync_t.rx.fifo.af

This field configures the Rx FIFO Almost Full threshold level. When applying a setting the Rx FIFO is reset and the current content is lost. The Rx FIFO Almost Full status is asserted when the Rx FIFO can receive *Almost Full* or fewer data values before becoming full.

Valid Values

Valid values are those within the inclusive range of zero to 0xFFFF. If a service is being used which applies a setting, then the value `-1` can be used to retrieve the current setting without applying or changing the current setting.

Low Level Function

```
int sio4_fasync_t_rx_fifo_af(
        int                     fd,
        s32*                    arg,
        sio4_fasync_action_t    action,       section 4.3.2, page 39
        FILE*                   file,
        sio4_fasync_error_t*    error);       section 4.3.3, page 39
```

The function argument and return values are described in section 4.1, page 34.

Calling Restrictions

There are no restrictions on calling this function.

4.3.5.45. sio4_fasync_t.rx.fifo.bytes

This field reports the number of bytes in the Rx FIFO. This is a read-only feature.

Valid Values

Valid values returned are those within the inclusive range of zero up through the size of the Rx FIFO.

Low Level Function

```
int sio4_fasync_t_rx_fifo_bytes(
        int                     fd,
        s32*                    arg,
        sio4_fasync_action_t    action,     section 4.3.2, page 39
        FILE*                   file,
        sio4_fasync_error_t*    error);     section 4.3.3, page 39
```

The function argument and return values are described in section 4.1, page 34.

Calling Restrictions

There are no restrictions on calling this function.

4.3.5.46. sio4_fasync_t.rx.fifo.full_cfg

This field configures the receiver's response to the Rx FIFO becoming full.

Valid Values

Valid options are those from the table below. If a service is being used which applies a setting, then the value `-1` can be used to retrieve the current setting without applying or changing the current setting.

Value	**Description**
`SIO4_FASYNC_RX_FIFO_FULL_CFG_DISABLE`	With this option the receiver will be disabled.
`SIO4_FASYNC_RX_FIFO_FULL_CFG_ENABLE`	With this option the receiver will remain enabled.

Low Level Function

```
int sio4_fasync_t_rx_fifo_full_cfg(
        int                     fd,
        s32*                    arg,
        sio4_fasync_action_t    action,       section 4.3.2, page 39
        FILE*                   file,
        sio4_fasync_error_t*    error);       section 4.3.3, page 39
```

The function argument and return values are described in section 4.1, page 34.

Calling Restrictions

Making a feature selection, even if it is not being changed, must be done only while the receiver is disabled. There are otherwise no restrictions on calling this function.

4.3.5.47. sio4_fasync_t.rx.fifo.overflow

This field pertains to the Rx FIFO overflow status. The service will always return the current overflow status.

Valid Values

Valid options are those from the table below. If a service is being used which applies a setting, then the value `-1` can be used to retrieve the current setting without applying or changing the current setting.

Value	**Description**
`SIO4_FASYNC_FIFO_OVERFLOW_CLEAR`	This option is used to clear the overflow status.
`SIO4_FASYNC_FIFO_OVERFLOW_TEST`	This option is used to request the overflow status.
`SIO4_FASYNC_FIFO_OVERFLOW_NO`	This option is returned when the current status indicates that an overflow has not occurred.
`SIO4_FASYNC_FIFO_OVERFLOW_YES`	This option is returned when the current status indicates that an overflow has occurred.

Low Level Function

```
int sio4_fasync_t_rx_fifo_overflow(
        int                     fd,
        s32*                    arg,
        sio4_fasync_action_t    action,         section 4.3.2, page 39
        FILE*                   file,
        sio4_fasync_error_t*    error);         section 4.3.3, page 39
```

The function argument and return values are described in section 4.1, page 34.

Calling Restrictions

There are no restrictions on calling this function.

4.3.5.48. sio4_fasync_t.rx.fifo.size

This field reports the size the Rx FIFO in bytes. This is a read-only feature.

Valid Values

Valid values returned should be within the inclusive range from 512 to 32,768.

Low Level Function

```
int sio4_fasync_t_rx_fifo_size(
        int                     fd,
        s32*                    arg,
        sio4_fasync_action_t    action,       section 4.3.2, page 39
        FILE*                   file,
        sio4_fasync_error_t*    error);       section 4.3.3, page 39
```

The function argument and return values are described in section 4.1, page 34.

Calling Restrictions

There are no restrictions on calling this function.

4.3.5.49. sio4_fasync_t.rx.fifo.status

This field reports the relative fill level of the Rx FIFO. This is a read-only feature.

Valid Values

Valid options returned are those from the table below.

Value	**Description**
`SIO4_FASYNC_FIFO_STATUS_EMPTY`	The FIFO is empty.
`SIO4_FASYNC_FIFO_STATUS_AE`	The FIFO is at or below the Rx FIFO Almost Empty level (section 4.3.5.43, page 85), though it is not empty.
`SIO4_FASYNC_FIFO_STATUS_MEDIUM`	The FIFO fill level is between the Rx FIFO Almost Empty and Rx FIFO Almost Full levels.
`SIO4_FASYNC_FIFO_STATUS_AF`	The FIFO is at or above the Rx FIFO Almost Full level (section 4.3.5.44, page 86), though it is not full.
`SIO4_FASYNC_FIFO_STATUS_FULL`	The FIFO is full.

Low Level Function

```
int sio4_fasync_t_rx_fifo_status(
        int                     fd,
        s32*                    arg,
        sio4_fasync_action_t    action,      section 4.3.2, page 39
        FILE*                   file,
        sio4_fasync_error_t*    error);      section 4.3.3, page 39
```

The function argument and return values are described in section 4.1, page 34.

Calling Restrictions

There are no restrictions on calling this function.

4.3.5.50. sio4_fasync_t.rx.fifo.underflow

This field pertains to the Rx FIFO underflow status. The service will always return the current underflow status.

Valid Values

Valid options are those from the table below. If a service is being used which applies a setting, then the value `-1` can be used to retrieve the current setting without applying or changing the current setting.

Value	**Description**
`SIO4_FASYNC_FIFO_UNDERFLOW_CLEAR`	This option is used to clear the underflow status.
`SIO4_FASYNC_FIFO_UNDERFLOW_TEST`	This option is used to request the underflow status.
`SIO4_FASYNC_FIFO_UNDERFLOW_NO`	This option is returned when the current status indicates that an underflow has not occurred.
`SIO4_FASYNC_FIFO_UNDERFLOW_YES`	This option is returned when the current status indicates that an underflow has occurred.

Low Level Function

```
int sio4_fasync_t_rx_fifo_overflow(
        int                     fd,
        s32*                    arg,
        sio4_fasync_action_t    action,      section 4.3.2, page 39
        FILE*                   file,
        sio4_fasync_error_t*    error);      section 4.3.3, page 39
```

The function argument and return values are described in section 4.1, page 34.

Calling Restrictions

There are no restrictions on calling this function.

4.3.5.51. sio4_fasync_t.rx.io.dma_threshold

This field specifies the minimum size of Block Mode DMA transfers performed by the driver. The device driver’s read service considers the size of the read request and the size of individual DMA requests before proceeding with a Block Mode DMA transfer. If the Rx FIFO has sufficient data to satisfy the read request, then the driver proceeds. Otherwise, if the size of the individual DMA request is below this threshold, then the driver will wait 1ms for the Rx FIFO to receive more data before trying again.

> **NOTE:** The driver may break individual Block Mode DMA read requests into multiple, smaller Block Mode DMA transfers. This is based on the size of the read request and the amount of available data in the Rx FIFO at the moment the Rx FIFO state is examined.

Valid Values

Valid values are the inclusive range from zero to the size of the Rx FIFO. If a service is being used which applies a setting, then the value `-1` can be used to retrieve the current setting without applying or changing the current setting.

Low Level Function

```
int sio4_fasync_t_rx_io_dma_threshold(
        int                     fd,
        s32*                    arg,
        sio4_fasync_action_t    action,      section 4.3.2, page 39
        FILE*                   file,
        sio4_fasync_error_t*    error);      section 4.3.3, page 39
```

The function argument and return values are described in section 4.1, page 34.

Calling Restrictions

There are no restrictions on calling this function.

4.3.5.52. sio4_fasync_t.rx.io.mode

This field selects the data transfer mode to be used for read requests.

Valid Values

Valid options are those from the table below. These macros are defined in the header file `gsc_common.h`. If a service is being used which applies a setting, then the value `-1` can be used to retrieve the current setting without applying or changing the current setting.

Value	**Description**
`GSC_IO_MODE_BMDMA`	This option refers to Block Mode DMA. †
`GSC_IO_MODE_DMDMA`	This mode refers to Demand Mode DMA. †
`GSC_IO_MODE_PIO`	This mode refers to PIO.

† The SIO4 has only two DMA engines. A BMDMA or DMDMA transfer request may fail if both DMA engines are already in use by other SIO4 I/O requests.

Low Level Function

```
int sio4_fasync_t_rx_io_mode(
        int                     fd,
        s32*                    arg,
        sio4_fasync_action_t    action,       section 4.3.2, page 39
        FILE*                   file,
        sio4_fasync_error_t*    error);       section 4.3.3, page 39
```

The function argument and return values are described in section 4.1, page 34.

Calling Restrictions

There are no restrictions on calling this function.

4.3.5.53. sio4_fasync_t.rx.io.overflow

This field specifies if the driver's read service should account for Rx FIFO Overflow conditions. The overflow check, when enabled, is performed upon entry to the driver's read service.

Valid Values

Valid options are those from the table below. If a service is being used which applies a setting, then the value `-1` can be used to retrieve the current setting without applying or changing the current setting.

Value	**Description**
`SIO4_FASYNC_IO_OVERFLOW_CHECK`	The driver checks for Rx FIFO overflows. †
`SIO4_FASYNC_IO_OVERFLOW_IGNORE`	The driver does not check for Rx FIFO overflows.

† If an overflow is detected, then the read request immediately returns an error status.

Low Level Function

```
int sio4_fasync_t_rx_io_overflow(
        int                     fd,
        s32*                    arg,
        sio4_fasync_action_t    action,      section 4.3.2, page 39
        FILE*                   file,
        sio4_fasync_error_t*    error);      section 4.3.3, page 39
```

The function argument and return values are described in section 4.1, page 34.

Calling Restrictions

There are no restrictions on calling this function.

4.3.5.54. sio4_fasync_t.rx.io.pio_threshold

This field specifies the threshold below which individual DMA based read requests and individual DMA transfer requests will instead use PIO. This mode change is made because very small DMA requests can be performed more quickly by using PIO.

> **NOTE:** The driver redirects individual DMA read requests and individual DMA transfer requests to use PIO when the volume of such requests is less than the configured threshold.

Valid Values

Valid values are the inclusive range from zero to 0xFFFFFFF. A value of zero disables the change to PIO. A value equal to or larger than the size of the Rx FIFO will essentially force all Block Mode DMA read requests to use PIO. A value equal to or larger than the 64K (an internal driver limit) will essentially force all Demand Mode DMA read requests to use PIO. If a service is being used which applies a setting, then the value `-1` can be used to retrieve the current setting without applying or changing the current setting.

Low Level Function

```
int sio4_fasync_t_rx_io_pio_threshold(
        int                     fd,
        s32*                    arg,
        sio4_fasync_action_t    action,     section 4.3.2, page 39
        FILE*                   file,
        sio4_fasync_error_t*    error);     section 4.3.3, page 39
```

The function argument and return values are described in section 4.1, page 34.

Calling Restrictions

There are no restrictions on calling this function.

4.3.5.55. sio4_fasync_t.rx.io.timeout

This field specifies the maximum amount of time, in seconds, that the driver permits for read requests. The driver's read service will return either when the request has been satisfied or when the timeout period expires, whichever occurs first.

Valid Values

Valid options are those in the range from the minimum to the maximum, as shown below, plus the *infinite* option. These macros are defined in header file `sio4.h`. If a service is being used which applies a setting, then the value `-1` can be used to retrieve the current setting without applying or changing the current setting.

Value	**Description**
`SIO4_IO_TIMEOUT_DEFAULT`	This is the default, which is 10 seconds.
`SIO4_IO_TIMEOUT_INFINITE`	This tells the driver to never timeout.
`SIO4_IO_TIMEOUT_MAX`	This is the maximum amount of time permitted, which is 3,600 seconds, or one hour.
`SIO4_IO_TIMEOUT_MIN`	This is the minimum, which is zero, and is the same as the *No Sleep* option.
`SIO4_IO_TIMEOUT_NO_SLEEP`	This tells the driver not to sleep to wait for more data in the Rx FIFO. In this case, the service returns rather than wait for more data.

Low Level Function

```
int sio4_fasync_t_rx_io_timeout(
        int                     fd,
        s32*                    arg,
        sio4_fasync_action_t    action,      section 4.3.2, page 39
        FILE*                   file,
        sio4_fasync_error_t*    error);      section 4.3.3, page 39
```

The function argument and return values are described in section 4.1, page 34.

Calling Restrictions

There are no restrictions on calling this function.

4.3.5.56. sio4_fasync_t.rx.io.underflow

This field specifies if the driver's read service should account for Rx FIFO underflow conditions. The underflow check, when enabled, is performed upon entry to the driver's read service.

Valid Values

Valid options are those from the table below. If a service is being used which applies a setting, then the value `-1` can be used to retrieve the current setting without applying or changing the current setting.

Value	**Description**
`SIO4_FASYNC_IO_UNDERFLOW_CHECK`	The driver checks for Rx FIFO underflows. †
`SIO4_FASYNC_IO_UNDERFLOW_IGNORE`	The driver does not check for Rx FIFO underflows.

† If an underflow is detected, then the read request immediately returns an error status.

Low Level Function

```
int sio4_fasync_t_rx_io_underflow(
        int                     fd,
        s32*                    arg,
        sio4_fasync_action_t    action,         section 4.3.2, page 39
        FILE*                   file,
        sio4_fasync_error_t*    error);         section 4.3.3, page 39
```

The function argument and return values are described in section 4.1, page 34.

Calling Restrictions

There are no restrictions on calling this function.

5. Operating Information

This section explains some basic operational procedures for using the SIO4 with the Fast Async Protocol Library. This is in no way intended to be a comprehensive guide. This is simply to address a very few issues relating to its use.

5.1. Getting Started

To configure the SIO4 for Fast Async operation meeting specific requirements, the recommended starting point is a local, customizable copy of the `fasyncc2c` sample application included with the driver. This application is designed to transfer bulk data between a Fast Async transmitter and a Fast Async receiver. The transmitter and receiver can be from two channels on different boards, two channels on the same board, or by loopback mode using the transmitter and receiver from the very same channel. There are two loopback configurations. Internal Loopback performs the transfer by routing signals internal to the firmware. External Loopback performs the transfer by routing the signals through the cable transceivers.

> **NOTE**: When using Loopback operation, it is recommended that cabling and any remote equipment be disconnected from the SIO4. This is because External Loopback is the default when Internal Loopback is not supported by firmware.

5.1.1. Cable Validation

The first step in deriving a customized configuration is to verify cabling. This should be done using the application's default settings. It is recommended that one channel be used for loopback testing and that two other channels on the same board be connected by the cabling to be tested. A script with the below commands is a convenient means of repeating these tests until cabling has been verified successfully.

```
./fasyncc2c 0 0 -i
./fasyncc2c 0 0 -e
./fasyncc2c 1 2
```

5.1.2. Customizing the Configuration

The second step in deriving a customized configuration is to methodically modify the application code, one parameter at a time, until all necessary parameter changes have been accommodated. That is, choose a parameter from the documentation that must be changed from its default, modify the application so the required setting can be specified from the command line, then test the resulting changes. A script with the below commands is a convenient means of repeating these tests. In this example the "`-X`" represents the new command line argument for the parameter being altered from its default. This script tests all three cabling setups both without the change and with the change. This is done to verify that the default operation remains functional after the code modifications. In some cases, the script may need to be expanded to test each parameter addition to ensure prior changes remain functional. It is suggested that parameter changes be accommodated one at a time to ease the development and testing process.

```
./fasyncc2c 0 0 -i
./fasyncc2c 0 0 -e
./fasyncc2c 1 2 -i
./fasyncc2c 0 0 -i -X
./fasyncc2c 0 0 -e -X
./fasyncc2c 1 2    -X
```

> **NOTE**: At times modifications for a parameter may need to be implemented on the transmitter or receiver first in order to facilitate validation. At other times the transmitter and receiver may temporarily be configured differently in order to verify that a change is implemented properly.

> **NOTE**: It is best to initially test parameter additions separately, one at a time. Where there are parameter interactions, testing parameter combinations should be completed before moving on.

The general sequence for addition of a new parameter modification is as follows.

1. Add a field for the new parameter to the `args_t` structure defined in `main.h`.

2. Add command line support for the new parameter by updating the `_parse_args()` function at the top of `main.c`. At minimum, the `list[]` table must be updated to associate assignment of a setting with a command line argument. This may also mean assigning a default to the new field following the `memset()` function call.

> **NOTE**: One may have to review multiple sample applications to get a feel for how to add specific command line argument types to the argument table.

3. Update the `_setup_apply()` function at the top of `setup.c` to apply the value from the new `args_t` field to the `sio4_fasync_t` structure prior to calling `sio4_fasync_set()`.

4. Now update the script steps given above for the code changes just implemented. Continue adding support for other required parameter changes when testing is complete. Update the above script for each addition.

5.2. Debugging Aids

The SIO4 driver archive includes the following debugging aids appropriate for use with the Fast Async Protocol Library.

5.2.1. Device Identification

When communicating with technical support complete device identification is virtually always necessary. The *id* example application is provided for this specific purpose. This is a text only console application. The output can be piped to a file, which can then be emailed to GSC technical support when requested. Locate the application as follows.

Description	**File**	**Location**
Application	`id`	`…/id/`

5.2.2. sio4_fasync_show()

The function `sio4_fasync_show()` (section 4.1.14, page 27) is part of the protocol library interface. The purpose of this function is to produce a human readable report of all fields included in the `sio4_fasync_t` structure (section 4.3.5, page 40) passed in as a function argument. The function is best used to report the structure's content both before it is passed to `sio4_fasync_set()` (section 4.1.13, page 22) and after it is passed to `sio4_fasync_get()` (section 4.1.2, page 13). The output can be used to help visualize the channel configuration reflected by the structure content. When used in conjunction with `sio4_fasync_set()`, the `sio4_fasync_show()` output indicates the state that `sio4_fasync_set()` is expected to produce. When used in conjunction with `sio4_fasync_get()`, the `sio4_fasync_show()` output indicates the channel's current state. This may be beneficial after calling `sio4_fasync_set()` in order to verify the results achieved. The pair of calls may also be used before or after the read or write I/O calls in order to help explain the results of individual transfer requests.

5.2.3. Detailed Register Dump

The function `sio4_reg_list()` is included in the SIO4 utility library. The purpose of the function is to report the current content of registers for the referenced serial channel. The arguments control the set of registers included in the output and the detail with which the register content is reported. This function can be called at any time to report the device state, but it is most often called after completing board setup, or just before or after `sio4_fasync_read()` or `sio4_fasync_write()` calls in order to help explain the results of individual transfer requests.

Prototype

```
int sio4_reg_list(int fd, int gsc, int gsc_detail,
        int usc, int usc_detail);
```

Argument	**Description**
`fd`	This is a file descriptor obtained from `sio4_fasync_open()` (section 4.1.8, page 20).
`gsc`	If non-zero, then the output will include a dump of all `GSC_SIO4_xxx` registers. Refer to `sio4.h` for a complete list of these registers.
`gsc_detail`	If non-zero, then the dump of the GSC registers will include detailed information about all register fields, including the field value and the meaning of the value.
`usc`	For Fast Async boards pass in a value of zero.
`usc_detail`	For Fast Async boards pass in a value of zero.

Return Value	**Description**
`>= 0`	This is the number of errors encountered during execution of the function.

5.2.4. Status Return Values

The Fast Async Protocol Library, the SIO4 API Library and the SIO4 device driver all report the results for each of the various interface services. The table below lists the most common error status values reported to an application.

Value	**`errno.h` Macro**	**Description**
`> 0`	None	This applies to read and write operations and reflects the number of bytes successfully transferred.
`0`	None	The operation was completed successfully.
`-5`	`-EIO`	The most likely cause is an Rx FIFO Overflow, though a Tx FIFO Overflow or a Rx FIFO Underflow also produce this result. An Rx FIFO Overflow error generally means the application isn’t reading data fast enough. The Tx FIFO errors should never happen, except under dedicated test conditions.
`-16`	`-EBUSY`	An open request failed due to a conflicting `share` argument. This can happen if an *Exclusive* open request is made when some application already has access to the same device. This will occur when the existing access is either *Shared* or *Exclusive*. This can also happen if a Shared request is made when the existing access is *Exclusive*. This status may also be returned by the close service if another API call is still active.
`-22`	`-EINVAL`	A function argument or referenced structure field value is invalid.
`-71`	`-EPROTO`	The Fast Async Protocol Library has not been initialized. Applications must call `sio4_fasync_init_api()` before making any other Library call.
`-77`	`-EBADF`	The file descriptor argument was not recognized. This indicates that the file descriptor is invalid, was not obtained by the `sio4_fasync_open()` function, or access to the referenced device has already been closed.
`-93`	`-EPROTONOSUPPORT`	This indicates that the SIO4 device doesn’t support the operation requested. This occurs when an API function is called on an SIO4 that is not based on the Fast Async firmware.

5.3. Cable Configuration Modes

Overall, the SIO4 supports three cable interfacing modes; DCE, DTE and Legacy mode. Older boards support only Legacy mode. More recent boards support only the DCE and DTE modes. Intermediate boards support all three modes. The Fast Async firmware supports the DCE and DTE mode selections only. The legacy mode is described in older SIO4 board user manuals.

5.4. Error and Status Detection

The SIO4 incorporates the ability to detect a number of error and other conditions for both the transmitter and the receiver.

5.4.1. Interrupt Events

One means of catching the various conditions, especially errors, is by use of interrupts. The basic steps for this are to enable the interrupts of interest then have a thread wait for a corresponding Wait Event request. This is illustrated in the following code fragments.

Thread A

```
For (;;)
{
      …
      read SIO4 data

      if (error recorded)
      {
            Error exists in
            1) Read buffer, or
            2) SIO4 Rx FIFO
            Resync data stream.
      }
      else
      {
            Read buffer is error free.
      }
      …
}
```

Thread B

```
For (;;)
{
      …
      Enable desired interrupts.
      Wait for an interrupt.

      if (error interrupt occurred)
      {
            Record the error.
      }
      …
}
```

5.5. Preloading Tx Data

As data is written to the SIO4 the Tx FIFO fill level can vary dramatically. Depending on the Tx bitrate, how rapidly an application is able to feed data to the transmitter, and various other factors, the Tx FIFO fill level may constantly hover around empty. As a result, the Tx FIFO may frequently, though inadvertently, become empty. If the transmitter is configured to be disabled when it becomes empty (section 4.3.5.28, page 69), then the inadvertent empty states can disable the transmitter prematurely. To prevent this from happening, the application may need to preload data into the Tx FIFO before having the data transmitted. This can be accomplished with the below pseudo code.

1. Configure the device up front with the *init_data* and *set* API functions.

2. Check to see how much data is in the Tx FIFO using the *Tx FIFO Bytes* function (section 4.3.5.27, page 68). If there is sufficient data in the Tx FIFO to prevent an inadvertent and premature empty state, then skip steps three through seven. What constitutes “sufficient data” may have to be determined experimentally.

3. Wait for the Tx FIFO to become empty by polling the *Tx FIFO Status* function (section 4.3.5.31, page 72). Inserting a millisecond sleep between successive calls may be appropriate.

4. Pause transmission by disabling the transmitter (section 4.3.5.20, page 61). So long as the transceivers are enabled, the TxD and RTS lines are driven to their idle states while the transmitter is disabled. While the transmitter should already be configured to disable the transmitter due to a Tx FIFO empty state, the application can request this via the API here as the Library will wait for the last byte to complete transmission.

5. The transmitter should now be disabled.

6. Write the necessary data to the Tx FIFO using the *write* API function (section 4.1.21, page 33), up to the size of the Tx FIFO.

7. Resume transmission by enabling the transmitter (section 4.3.5.20, page 61).

8. Write the remaining data to the TX FIFO using the *write* API function (section 4.1.21, page 33)

5.6. Waiting for the Transmitter or Receiver to Finish

There are times when an application may need to wait for transmission or reception to complete before continuing.

For the transmitter, the simplest case is waiting for the current byte, if any, to finish. If so, then the application should disable the transmitter (section 4.3.5.20, page 62). Afterwards, the application should poll the Tx Status function (section 4.3.5.24, page 66) till the status is reported as idle. At this point the transmitter has finished.

If the transmitter must send out all data in the Tx FIFO, then the application must monitor the Tx FIFO state then Tx Status for completion. The first step can be done by polling the Tx FIFO Fill Level (section 4.3.5.27, page 69), polling the Tx FIFO Status (section 4.3.5.31, page 73) or using a Wait Event request (section 4.1.19, page 32) for a Tx FIFO Empty interrupt. Once the Tx FIFO is empty, the application should then monitor the transmitter to determine when the last byte has been sent out the cable interface. This can be done by calling the Tx Status function (section 4.3.5.24, page 66) until the status is reported as idle. At this point the transmitter has finished.

> **NOTE:** There may be a slight delay between the Tx FIFO reporting its empty state and when the transmitter reports that it is busy sending out that last byte. Application should therefore initially read the transmitter status twice in succession after the Tx FIFO becomes empty.

For the receiver, the simplest case is waiting for the current byte, if any, to finish. If so, then the application should disable the receiver (section 4.3.5.37, page 79). Afterwards, the application should poll the Rx Status function (section 4.3.5.49, page 91) till the status is reported as idle. At this point the receiver has finished.

If the receiver must receive a complete message (block or frame), then the application must monitor the received data to determine when the message is complete. At this point the receiver can be disabled (section 4.3.5.37, page 79). Being disabled, the receiver may have already started to receive another byte. Depending on the bitrate, it is possible that multiple bytes have already accumulated in the Rx FIFO. These can be read (section 4.1.9, page 22) from the Rx FIFO and discarded. At this point the receiver has finished.

Document History

Revision	**Description**
March 19, 2024	Initial release.