SIO4/8

Four/Eight Channel High Speed Serial I/O

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

Asynchronous 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 Asynchronous Protocol Library, which is a library designed to facilitate use of the Asynchronous serial protocol with an SIO4 or SIO8.

1.1. Purpose

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

1.2. Acronyms

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

Acronyms	Description
DMA	Direct Memory Access
GSC	General Standards Corporation
PCI	Peripheral Component Interconnect
PMC	PCI Mezzanine Card
USC	Universal Serial Controller

1.3. Definitions

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

Term	Definition
Application	Application means the user mode process, which runs in user space with user mode privileges.
ASYNC	This refers to the Asynchronous serial protocol.
Driver	Driver means the executable providing the direct access to the SIO4 hardware.
Library	Depending on context, this is a general reference to the Asynchronous Protocol Library.
SIO4	This is used as a general reference to any Zilog based 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 SIO4A or SIO4B.

1.4. Software Overview

The Asynchronous Protocol Library is a statically linked library providing an Asynchronous centric interface to the SIO4 device driver. The library is provided in source form and must be built before being used. The library is a thin software layer that sits between an SIO4 application and the SIO4 API Library. The interface provided by the library is Asynchronous specific and is a simplified rendition of the IOCTL services that are part of the overall driver interface. The library exists in parallel with the driver interface.

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 two computers, or one computer and an external peripheral. Once the data link between the two devices is established, the desired transfers can be performed and will become transparent to the user. The SIO4 board includes two DMA controllers and comes with a maximum of 256K Bytes of FIFO storage, which is 32K per channel direction (32K * 2 * 4). Each DMA controller is capable of transferring data to and from host memory; whereas the FIFO helps maintain continuous data transfer at the cable interface. The FIFO configuration can vary greatly from one SIO4 version to another (i.e., 32K * 2 * 4 to 1K * 2 *1 to none at all). The SIO4 comes with

transceivers that are fixed as RS232 or RS485/422, or with transceivers that are configurable. The SIO4 comes in two basic varieties; SYNC models or Zilog models, which are based on two Z16C30 dual USC chips. Later model SIO4 boards support both models with the mode being software controlled on a per channel basis. The SIO4 also provides for interrupt generation for various states of the board like Sync Character detection, FIFO empty, FIFO full and DMA complete.

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

1.6. Reference Material

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

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

- The *Z16C30 USC User's Manual* from Zilog.

  Zilog, Inc.
  910 E Hamilton Ave
  Campbell, California 95008 USA
  Phone: 1-408-558-8500
  WEB: http://www.zilog.com/

2. The Asynchronous Serial Protocol

2.1. Description

The Asynchronous serial communications protocol is a byte oriented serial transmission protocol consisting of a Start Bit, a number of data bits, an optional Parity Bit, and up to 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. From this point the line level is sampled at about the center of each bit period. At the center of the Start Bit period, the line level is examined. If it is high, then the high to low transition is ignored as noise, and the decoding logic returns to waiting for another high-to-low transition. If the line is still low (the Space state, which is a 0 bit) then decoding continues per the byte size, use of parity and at least ½ Stop Bit. 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 the Stop Bit is sampled, then the result is a Framing Error.

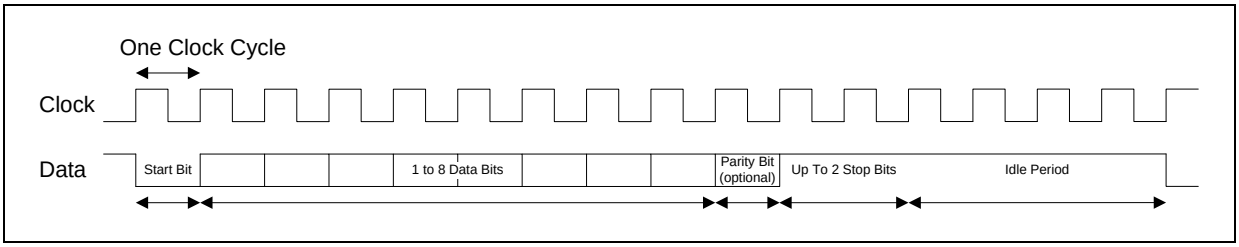

Figure 1 A depiction of an Asynchronous data stream.

The minimum signals needed for full-duplex communication are TxD and RxD (*Transmit Data* and *Receive Data*). Hardware flow control requires additional signals. The RTS/CTS flow control mechanism requires two additional signals. (RTS refers to *Request To Send* and CTS refers to *Clear To Send*.) The DTR/DSR flow control mechanism also requires two signals. (DTR refers to *Data Terminal Ready* and DST refers to *Data Set Ready*.) Communications with a modem may also use the DCD signal (DCD refers to *Data Carrier Detect*).

Equipment using the Asynchronous protocol usually uses the RS-232 line protocol. The number of signals varies with individual implementations and the set of supported signals. The arrangement of the supported signals is typically DCE or DTE, or some other custom implementation. (DCE refers to *Data Communications Equipment* and DTE refers to *Data Terminal Equipment*.) The typical upper baud rate limit is 115,200 bits-per-second. Higher rates are possible with special RS-232 transceivers or by using other than RS-232.

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 from just over ½ bit period to two-bit periods.

3. Library Interface Files

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

3.1. Header File

The library's interface is defined via the header file shown below. To use the Asynchronous 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_async.h`	`…/sio4/include`

3.2. Static Library Files

The executable code for the API defined for the Asynchronous Protocol Library is contained in the static library file `sio4_async.a`, which is 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 Asynchronous 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
Asynchronous Protocol Library	`sio4_async.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_async.h`, which is located in the `…/sio4/include/` directory.

4.1. High Level Functions

The library header defines the complete Asynchronous interface offered by the library, which includes the following high level function declarations.

4.1.1. sio4_async_close()

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

Prototype

```
int sio4_async_close(int fd);
```

Argument	Description
`fd`	This is a file descriptor obtained from `sio4_async_open()` (section 4.1.6, page 12).

Return Value	Description
`0`	The operation completed successfully.
`< 0`	An error occurred. This is a negative `errno.h` value.

4.1.2. sio4_async_get()

This function 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_async_t` structure.

Prototype

```
int sio4_async_get(int fd, sio4_async_t* async, const char** err);
```

Argument	Description
`fd`	This is a file descriptor obtained from `sio4_async_open()` (section 4.1.6, page 12).
`async`	This is the structure where the current settings are recorded (section 4.3.1, page 16). Any field pertaining to an unsupported feature will be set to `-1`. The value `-2` indicates a hardware setting that is invalid.
`err`	In the event of an error this will be set to identify the source of the error. The caller may provide a NULL pointer.

Return Value	Description
`0`	The operation completed successfully.
`< 0`	An error occurred. This is a negative `errno.h` value.

4.1.3. sio4_async_init()

This function initializes the Asynchronous Protocol Library and must be the first call into the library.

NOTE: This service initializes the Asynchronous 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 library. Any subsequent call has no effect.

Prototype

```
int sio4_async_init(void);
```

Argument	Description
None	The function has no arguments.

Return Value	Description
0	The operation completed successfully.
< 0	An error occurred. This is a negative errno.h value.

4.1.4. sio4_async_init_data()

This function initializes an sio4_async_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_async_t structure.

Prototype

```
int sio4_async_init_data(
        int                         fd,
        const sio4_async_init_t*    init,
        sio4_async_t*               async,
        const char**                err);
```

Argument	Description
fd	This is a file descriptor obtained from sio4_async_open() (section 4.1.6, page 12).
init	This structure provides the basic information needed prior to initializing the next argument. See below for more information.
async	This is the structure that the call will initialize (section 4.3.1, page 16).
err	In the event of an error this will be set to identify the source of the error. The caller may provide a NULL pointer.

Return Value	Description
0	The operation completed successfully.
< 0	An error occurred. This is a negative errno.h value.

Data Type

This structure contains information used to configure Asynchronous clocking for the USC transmitter and receiver.

NOTE: For additional information refer to Clocking Configuration (section 5.3.4, page 41), which gives information on initializing this structure.

```
typedef struct
{
    s32 tx_bit_rate;
    s32 rx_bit_rate;
```

```
    s32 osc_prog;
} sio4_async_init_t;
```

Field	Description
tx_bit_rate	This is the desired bit rate for the transmitter. This value must be greater than or equal to one, and less than or equal to 1,250,000.
rx_bit_rate	This is the desired bit rate for the receiver. This value must be greater than or equal to one, and less than or equal to 1,250,000.
osc_prog	This is the frequency to which the channel's on-board oscillator is to be programmed. This should be the highest possible multiple of the product of the Tx or Rx bit rate multiplied by 16, 32 or 64, but no higher than 20,000,000 (i.e., osc_prog = 20000000 – (20000000 % (16 * tx_bit_rate))). The factor 16 may generally provide better results if the Tx and Rx baud rates differ or if the Rx baud rate must be generated from the USC's DPLL. If the on-board oscillator is not programmable, then this should be set to the frequency of the fixed on-board oscillator. The default rate is 20,000,000 Hz.

4.1.5. sio4_async_ioctl()

This function is the entry point to performing IOCTL operations on the device. Refer to the driver reference manual for complete information on the driver's set of IOCTL services.

Prototype

```
int sio4_async_ioctl(int fd, int request, void* arg);
```

Argument	Description
fd	This is a file descriptor obtained from sio4_async_open() (section 4.1.6, page 12).
request	This is an IOCTL macro contained in sio4.h or sio4_usc.h.
arg	This is the argument type required for the above referenced IOCTL service.

Return Value	Description
0	The operation completed successfully.
< 0	An error occurred. This is a negative errno.h value.

4.1.6. sio4_async_open()

This function is the entry point to open a connection to an SIO4 serial channel for Asynchronous operation. The handle returned by this call is used for all subsequent access to the specified channel. The file descriptor returned can be used for access the library's high-level functions, the library's low-level functions, and the driver interface.

NOTE: If the value of the index argument is specified as -1, then the function opens the file /proc/sio4 for reading. In this case the share argument is ignored.

NOTE: If the value of the index argument is specified as -1, then the file descriptor returned can be used only with sio4_async_read() (section 4.1.7, page 13) and sio4_async_close() (section 4.1.1, page 10). It may also be used with the SIO4 API functions sio4_read() and sio4_close() (refer to the *SIO4 Reference Manual*). The file descriptor cannot be used with any other services.

Prototype

```
int sio4_async_open(int index, int share, int* fd);
```

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. The pointer cannot be NULL. Values returned are as follows. File descriptors provided by this call can be used interchangeably with all functions provided by the Asynchronous Protocol Library and by the SIO4 API Library. The same cannot be said of file descriptors obtained by any other means.

Value	Description
>= 0	This is the handle to use to access the device in subsequent calls.
-1	There was an error. The device is not accessible.

Return Value	Description
0	The operation completed successfully.
< 0	An error occurred. This is a negative errno.h value.

4.1.6.1. 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.

4.1.7. sio4_async_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.

NOTE: All read requests are serialized.

Prototype

```
int sio4_async_read(int fd, void* buf, size_t bytes);
```

Argument	Description
fd	This is a file descriptor obtained from sio4_async_open() (section 4.1.6, page 12).
buf	The data read is placed here.
bytes	Read up to this number of bytes.

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. This is a negative errno.h value.

4.1.8. sio4_async_set()

This function configures an SIO4 channel according to the settings of the referenced sio4_async_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_async_t structure.

> **NOTE:** Before calling this function, the structure should, at minimum, be initialized by calling the sio4_async_init_data() function (section 4.1.4, page 10).

Prototype

```
int sio4_async_set(
        int                 fd,
        const sio4_async_t* async,
        const char**        err);
```

Argument	Description
fd	This is a file descriptor obtained from sio4_async_open() (section 4.1.6, page 12).
async	This is the structure containing the settings to be applied (section 4.3.1, page 16).
err	In the event of an error this will be set to identify the source of the error. The caller may provide a NULL pointer.

Return Value	Description
0	The operation completed successfully.
< 0	An error occurred. This is a negative errno.h value.

4.1.9. sio4_async_show()

This function displays the content of the referenced sio4_async_t structure to the screen. This is provided to assist debugging efforts. This function operates mostly by calling the low-level functions corresponding to each field of the sio4_async_t structure.

Prototype

```
int sio4_async_show(
        int                 fd,
        const sio4_async_t* async,
        FILE*               file,
        const char**        err);
```

Argument	Description
fd	This is a file descriptor obtained from sio4_async_open() (section 4.1.6, page 12).
async	This is the structure whose content will be displayed (section 4.3.1, page 16).
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.

err	In the event of an error this will be set to identify the source of the error. The caller may provide a NULL pointer.

Return Value	Description
0	The operation completed successfully.
< 0	An error occurred. This is a negative errno.h value.

4.1.10. sio4_async_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.

NOTE: All write requests are serialized.

Prototype

```
int sio4_async_write(int fd, const void* buf, size_t bytes);
```

Argument	Description
fd	This is a file descriptor obtained from sio4_async_open() (section 4.1.6, page 12).
buf	This is the source for the data to write.
bytes	Write at most this number of bytes.

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. This is a negative errno.h value.

4.2. Low Level Functions

The low-level functions provide access to the board features reflected by the individual fields of the sio4_async_t structure (section 4.3.1, page 16). This structure is used to access all of the board features that are part of the Asynchronous Protocol Library. 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_async_t structure, the high-level functions perform their work by calling the low-level functions for each 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_async_t structure fields. This is where an application can make use of the low-level function. All of the low-level functions follow the prototype pattern shown below. The function naming convention includes the prefix "sio4_async_t_" followed by the C style field name, but with the periods (".") replaced by underscores ("_").

NOTE: The low-level functions do not accept NULL pointers.

Prototype

```
void sio4_async_t_field_name(
    int                 fd,
    s32*                arg,
    sio4_async_action_t action,
    const char**        err);
```

Argument	Description
fd	This is a file descriptor obtained from sio4_async_open() (section 4.1.6, page 12).
arg	This refers to the feature specific value being passed in to the function. This pointer must not be NULL.
action	This identifies the specific action to be carried out in regards to the above feature specific value. See the sio4_async_action_t data type below.
err	If there is an error, then this field will be set to point to a string naming the sio4_async_t field for which the error pertains. An example is "async->cable.loopback.mode". This pointer must not be NULL.

Data Type

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

```
typedef enum
{
    SIO4_ASYNC_ACTION_GET,
    SIO4_ASYNC_ACTION_INIT,
    SIO4_ASYNC_ACTION_SET,
    SIO4_ASYNC_ACTION_SHOW,
    SIO4_ASYNC_ACTION_VERIFY
} sio4_async_action_t;
```

Value	Description
SIO4_ASYNC_ACTION_GET	This requests the current setting from the driver.
SIO4_ASYNC_ACTION_INIT	This requests the initialization value from the library.
SIO4_ASYNC_ACTION_SET	This requests that the supplied value be applied by the driver.
SIO4_ASYNC_ACTION_SHOW	This requests that the supplied value be displayed to the screen.
SIO4_ASYNC_ACTION_VERIFY	This requests that the supplied value be verified.

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 4.3.1, page 16.

Field	Function
sio4_async_t.cable.loopback.mode	sio4_async_t_cable_loopback_mode()
sio4_async_t.rx.encoding	sio4_async_t_rx_encoding()
sio4_async_t.tx.parity.enable	sio4_async_t_tx_parity_enable()

4.3. Data Structures

Including the library header (sio4_async.h) in a source file gives that source the full library and driver interface. The library header defines the complete Asynchronous interface offered by the library, which includes the following data structures and their associated macros and low-level function declarations.

4.3.1. sio4_async_t

This structure contains all of the parameters used to configure an SIO4 channel for Asynchronous operation. The structure is initialized with default values by calling the sio4_async_init_data() function (section 4.1.4,

page 10). Following this call, applications make changes to this structure's content according to their own requirements. Afterwards, the structure is passed to the `sio4_async_set()` function (section 4.1.8, page 14) where the settings are applied to the board.

```
typedef struct
{
    struct
    {
        s32     ref;
        s32     prog;
    } osc;

    Struct  // cable
    {
        s32     enable;     // PSRCR D31
        s32     mode;       // PSRCR D28, DCE or DTE
        s32     protocol;   // PSRSR D24-D27
        s32     txc;        // PSRCR D6-D8
        s32     txd;        // PSRCR D19-D20
        s32     txaux;      // PSRCR D17-D18
        s32     dcd;        // PSRCR D15-D16
        s32     dtr_dsr;    // PSRCR D21-D22
        s32     rts;        // PSRCR D13-D14

        struct
        {
            s32 mode;       // PSRCR D23, D29
        } loopback;

        struct
        {
            s32 enable;     // PSRCR D30
        } term;

        struct
        {
            s32 txc;        // CCR 0x3333
            s32 txd_cts;    // CSR D2-D3
            s32 rxc;        // CCR 0xCCCC
            s32 rxd_dcd;    // CSR D4-D5
        } legacy;

    } cable;

    Struct  // tx
    {
        s32     mode;       // USC CMR D8-D11
        s32     enable;     // USC TMR D0-D1
        s32     char_len;   // USC TMR D2-D4
        s32     encoding;   // USC TMR D13-D15
        s32     bit_rate;   // reflects sio4_async_init_t.tx_bit_rate
        s32     idle_cond;  // USC TCSR D8-D10

        struct
        {
            s32 enable;     // USC TMR D5
            s32 type;       // USC TMR D6-D7
```

```
    } parity;

    struct
    {
        s32 size;       // FSR D0-D15, read-only
        s32 ae;         // TAR D0-D15
        s32 af;         // TAR D16-D31
        s32 empty_cfg;  // CSR D18, D26
        s32 space_cfg;  // CSR D4-D5
    } fifo;

    struct
    {
        s32 dma_thresh; // Block Mode DMA only
        s32 mode;
        s32 pio_thresh;
        s32 timeout;
        s32 overrun;
    } io;

} tx;

Struct  // rx
{
    s32     mode;       // USC CMR D0-D3
    s32     enable;     // USC RMR D0-D1
    s32     char_len;   // USC RMR D2-D4
    s32     encoding;   // USC RMR D13-D15
    s32     bit_rate;   // reflects sio4_async_init_t.rx_bit_rate
    s32     sync_byte;  // SBR D0-D7
    s32     status_word;// CSR D3

    struct
    {
        s32 enable;     // USC RMR D5
        s32 type;       // USC RMR D6-D7
    } parity;

    struct
    {
        s32 size;       // FSR D16-D21, read-only
        s32 ae;         // RAR D0-D15
        s32 af;         // RAR D16-D31
        s32 full_cfg;   // BCR D8
    } fifo;

    struct
    {
        s32 dma_thresh; // Block Mode DMA only
        s32 mode;
        s32 pio_thresh;
        s32 timeout;
        s32 overrun;
        s32 underrun;
    } io;

    struct
```

```
    {
        s32 enable;     // CSR D2
        s32 clk_src;    // BCR D22
    } time_stamp;

} rx;

Struct  // usc
{
    s32     mode;       // USC CCAR D8-D9
    s32     txd;        // USC IOCR D6-D7
    s32     cts;        // PSRCR D9-D10, USC IOCR D14-D15
    s32     cts_legacy; // USC IOCR D14-D15
    s32     dcd;        // PSRCR D11-D12, USC IOCR D12-D13
    s32     dcd_legacy; // USC IOCR D12-D13

    // All of the folling USC fields are initialized
    // by sio4_async_init_data() based on the content of the
    // sio4_async_init_t structure.

    struct
    {
        s32 clk_rate;   // USC CMR D4-D5
        s32 clk_src;    // USC CMCR D3-D5
        s32 txc;        // PSRCR D0-D2, USC IOCR D3-D5
        s32 txc_legacy; // USC IOCR D3-D5
        s32 stop_bits;  // USC CMR D14
    } tx;

    struct
    {
        s32 clk_rate;   // USC CMR D12-D13
        s32 clk_src;    // USC CMCR D0-D2
        s32 rxc;        // PSRCR D3-D5, USC IOCR D0-D2
        s32 rxc_legacy; // USC IOCR D0-D2
    } rx;

    struct
    {
        s32 enable;     // USC HCR D0
        s32 clk_src;    // USC CMCR D8-D9
        s32 divider;    // USC TC1R D0-D15
        s32 mode;       // USC HCR D1
    } brg0;

    struct
    {
        s32 enable;     // USC HCR D4
        s32 clk_src;    // USC CMCR D10-D11
        s32 divider;    // USC TC0R D0-D15
        s32 mode;       // USC HCR D5
    } brg1;

    struct
    {
        s32 clk_src;    // USC CMCR D12-D13
        s32 rate;       // USC HCR D14-D15
```

```
            } ctr0;

            struct
            {
                s32 clk_src;    // USC CMCR D14-D15
                s32 rate_src;   // USC HCR D13, ...
            } ctr1;

            struct
            {
                s32 rate;       // USC HCR D10-D11
            } dpll;

        } usc;

    } sio4_async_t;
```

4.3.1.1. sio4_async_t.osc

This section describes the structure's oscillator configuration fields.

Field	Description
`osc`	This structure configures the oscillator interface.
`osc.ref`	This field specifies the frequency of the fixed onboard reference oscillator. The default is 20MHz. The feature's low-level function is `sio4_async_t_osc_ref()`. This value is provided for informational purposes only.
`osc.prog`	This field specifies the desired programmable oscillator frequency. This is essentially the clock frequency provided by the onboard programmable oscillator to the USC. The default is 20MHz. The feature's low-level function is `sio4_async_t_osc_prog()`. The value provided is not recorded for later retrieval. Retrieval requests return one if oscillator programming is supported and zero if it isn't.

4.3.1.2. sio4_async_t.cable

This section describes the structure's cable configuration fields.

Field	Description
`cable`	This structure configures the cable interface.
`cable.enable`	This field either enables or disables the cable transceivers. Valid values are given in the table below. The feature's low-level function is `sio4_async_t_cable_enable()`. Please also read Cable Configuration Modes (section 5.5, page 42).

Value	Description
`SIO4_ASYNC_CABLE_ENABLE_NO`	Disable the cable transceivers.
`SIO4_ASYNC_CABLE_ENABLE_YES`	Enable the cable transceivers. This is the default. This option disables all legacy cable related settings.

`cable.mode`	This field specifies the arrangement of the signals on the cable interface. Valid values are given in the table below. The feature's low-level function is `sio4_async_t_cable_mode()`. Please also read Cable Configuration Modes (section 5.5, page 42).

Value	Description
`SIO4_ASYNC_CABLE_MODE_DCE`	Select the DCE cable signal configuration.
`SIO4_ASYNC_CABLE_MODE_DTE`	Select the DTE cable signal configuration. This is the default.

cable.Protocol

This field specifies the cable transceiver configuration. The options available depend on the board's transceiver capabilities. Valid values are given in the table below. The feature's low-level function is `sio4_async_t_cable_protocol()`.

Value	Description
SIO4_ASYNC_CABLE_PROTOCOL_RS232	This selects the RS232 protocol. This is the default.
SIO4_ASYNC_CABLE_PROTOCOL_RS422_423_1	This selects the RS422/RS423 mixed protocol version 1.
SIO4_ASYNC_CABLE_PROTOCOL_RS422_423_2	This selects the RS422/RS423 mixed protocol version 2.
SIO4_ASYNC_CABLE_PROTOCOL_RS422_RS485	This selects the RS422/RS485 mixed protocol.
SIO4_ASYNC_CABLE_PROTOCOL_RS423	This selects the RS423 protocol.
SIO4_ASYNC_CABLE_PROTOCOL_RS530	This selects the RS530 protocol, version 1.
SIO4_ASYNC_CABLE_PROTOCOL_RS530A	This selects the RS530 protocol, version 2.
SIO4_ASYNC_CABLE_PROTOCOL_V35	This selects the V.35 protocol, version 1.
SIO4_ASYNC_CABLE_PROTOCOL_V35A	This selects the V.35 protocol, version 2.

cable.txc

This field specifies the configuration of the cable's Tx Clock signal. Valid values are given in the table below. The feature's low-level function is `sio4_async_t_cable_txc()`.

Value	Description
SIO4_ASYNC_CABLE_TXC_OUT_0	This drives the signal low.
SIO4_ASYNC_CABLE_TXC_OUT_1	This drives the signal high.
SIO4_ASYNC_CABLE_TXC_OUT_CBL_RXA	This drives the signal from what appears at the cable's Rx Aux signal.
SIO4_ASYNC_CABLE_TXC_OUT_CBL_RXC	This drives the signal from what appears at the cable's Rx Clock signal.
SIO4_ASYNC_CABLE_TXC_OUT_OSC	This drives the signal from the onboard oscillator.
SIO4_ASYNC_CABLE_TXC_OUT_OSC_INV	This drives the signal from the inverted form of the onboard oscillator.
SIO4_ASYNC_CABLE_TXC_OUT_USC_RXC	This drives the signal from what appears at the USC's Rx Clock pin.
SIO4_ASYNC_CABLE_TXC_OUT_USC_TXC	This drives the signal from what appears at the USC's Tx Clock pin. This is the default.

cable.txd

This field specifies the configuration of the cable's Tx Data signal. Valid values are given in the table below. The feature's low-level function is `sio4_async_t_cable_txd()`.

Value	Description
SIO4_ASYNC_CABLE_TXD_OUT_0	This drives the signal low.
SIO4_ASYNC_CABLE_TXD_OUT_1	This drives the signal high.
SIO4_ASYNC_CABLE_TXD_OUT_USC_TXD	This drives the signal from what appears at the USC's Tx Data pin. This is the default.

cable.txaux

This field specifies the configuration of the cable's Tx Aux signal. Valid values are given in the table below. The feature's low-level function is `sio4_async_t_cable_txaux()`.

Value	Description
SIO4_ASYNC_CABLE_TXAUX_OUT_0	This drives the signal low.
SIO4_ASYNC_CABLE_TXAUX_OUT_1	This drives the signal high.
SIO4_ASYNC_CABLE_TXAUX_OUT_OSC	This drives the signal from the onboard oscillator.
SIO4_ASYNC_CABLE_TXAUX_TRI	This tri-states the drive segment of the transceivers. This is the default.

cable.dcd

This field specifies the cable DCD signal source when the cable signal is driven. Valid values are given in the table below. The feature's low-level function is `sio4_async_t_cable_dcd()`.

NOTE: Refer to the `usc.dcd` field (section 4.3.1.5, page 30) for affecting the cable signal's driven state.

Value	Description
SIO4_ASYNC_CABLE_DCD_OUT_0	This drives the signal low.
SIO4_ASYNC_CABLE_DCD_OUT_1	This drives the signal high.
SIO4_ASYNC_CABLE_DCD_OUT_RTS	This drives the signal from the Rx FIFO Almost Full status. This is the default.
SIO4_ASYNC_CABLE_DCD_OUT_USC_DCD	This drives the signal from what appears at the USC's DCD pin.

cable.dtr_dsr

This field specifies the configuration of the cable's DTR/DSR signal. Valid values are given in the table below. The feature's low-level function is `sio4_async_t_cable_dtr_dsr()`.

Value	Description
SIO4_ASYNC_CABLE_DTR_DSR_OUT_0	This drives the signal low.
SIO4_ASYNC_CABLE_DTR_DSR_OUT_1	This drives the signal high.
SIO4_ASYNC_CABLE_DTR_DSR_IN	This configures the signal as an input.
SIO4_ASYNC_CABLE_DTR_DSR_TRI	This tri-states the drive segment of the transceivers. This is the default.

cable.rts

This field specifies the configuration of the cable's RTS signal. Valid values are given in the table below. The feature's low-level function is `sio4_async_t_cable_rts()`.

Value	Description
SIO4_ASYNC_CABLE_RTS_OUT_0	This drives the signal low.
SIO4_ASYNC_CABLE_RTS_OUT_1	This drives the signal high.
SIO4_ASYNC_CABLE_RTS_OUT_CTS	This drives the signal from what appears at the USC's CTS pin.
SIO4_ASYNC_CABLE_RTS_OUT_RTS	This drives the signal from the Rx FIFO Almost Full status. This is the default.

cable.loopback

This structure configures the cable's loopback feature.

cable.loopback.mode

This field specifies the loopback mode. Valid values are given in the table below. The feature's low-level function is `sio4_async_t_cable_loopback_mode()`.

Value	Description
SIO4_ASYNC_LOOPBACK_MODE_DISABLE	This disables loopback operation. This is the default.
SIO4_ASYNC_LOOPBACK_MODE_EXTERNAL	This selects the external loopback mode. *†
SIO4_ASYNC_LOOPBACK_MODE_INTERNAL	This selects the internal loopback mode. †

* If external loopback mode is requested but not available, then the internal loopback mode is selected.

† Both loopback modes are performed onboard the SIO4; internal inside the USC, external at the

<table>
<tr><td></td><td>cable interface. For external mode, enable the transceivers and disconnect cabling.</td></tr>
<tr><td>cable.
term</td><td>This structure configures the cable's termination feature. The operation of this feature depends on the active cable transceiver type.</td></tr>
<tr><td>cable.
term.
enable</td><td>This field specifies the configuration of the transceiver's built-in termination feature. Valid values are given in the table below. The feature's low-level function is sio4_async_t_cable_term_enable().
<table><tr><th>Value</th><th>Description</th></tr><tr><td>SIO4_ASYNC_CABLE_TERM_ENABLE_NO</td><td>The built-in termination is disabled. This is the default.</td></tr><tr><td>SIO4_ASYNC_CABLE_TERM_ENABLE_YES</td><td>The built-in termination is enabled.</td></tr></table></td></tr>
<tr><td>cable.
legacy</td><td>This structure configures the cable's legacy interface feature. These fields are utilized if the board DCE/DTE cable configuration feature is absent or unused. Please also read Cable Configuration Modes (section 5.5, page 42).</td></tr>
<tr><td>cable.
legacy.
txc</td><td>This field specifies the legacy configuration of the cable's Tx Clock signal. Valid values are given in the table below. The feature's low-level function is sio4_async_t_cable_legacy_txc(). Please also read Cable Configuration Modes (section 5.5, page 42).
<table><tr><th>Value</th><th>Description</th></tr><tr><td>SIO4_ASYNC_CABLE_LEGACY_TXC_DISABLE</td><td>This disables the Tx Clock signal.</td></tr><tr><td>SIO4_ASYNC_CABLE_LEGACY_TXC_BOTH</td><td>This drives the Tx Clock signal on both the upper and lower group of pins.</td></tr><tr><td>SIO4_ASYNC_CABLE_LEGACY_TXC_LOW</td><td>This drives the Tx Clock signal on the lower group of pins.</td></tr><tr><td>SIO4_ASYNC_CABLE_LEGACY_TXC_UP</td><td>This drives the Tx Clock signal on the upper group of pins. This is the default.</td></tr></table></td></tr>
<tr><td>cable.
legacy.
txd_cts</td><td>This field specifies the legacy configuration of the cable's Tx Data and CTS signals. Valid values are given in the table below. The feature's low-level function is sio4_async_t_cable_legacy_txd_cts(). Please also read Cable Configuration Modes (section 5.5, page 42).
<table><tr><th>Value</th><th>Description</th></tr><tr><td>SIO4_ASYNC_CABLE_LEGACY_TXD_CTS_BOTH</td><td>This drives the signals on both the upper and lower group of pins.</td></tr><tr><td>SIO4_ASYNC_CABLE_LEGACY_TXD_CTS_LOW</td><td>This drives the signals on the lower group of pins.</td></tr><tr><td>SIO4_ASYNC_CABLE_LEGACY_TXD_CTS_TRI</td><td>This tri-states the signals.</td></tr><tr><td>SIO4_ASYNC_CABLE_LEGACY_TXD_CTS_UP</td><td>This drives the signals on the upper group of pins. This is the default.</td></tr></table></td></tr>
<tr><td>cable.
legacy.
rxc</td><td>This field specifies the legacy configuration of the cable's Rx Clock signal. Valid values are given in the table below. The feature's low-level function is sio4_async_t_cable_legacy_rxc(). Please also read Cable Configuration Modes (section 5.5, page 42).
<table><tr><th>Value</th><th>Description</th></tr><tr><td>SIO4_ASYNC_CABLE_LEGACY_RXC_DISABLE</td><td>This disables the Rx Clock signal.</td></tr><tr><td>SIO4_ASYNC_CABLE_LEGACY_RXC_LOW</td><td>This drives the Rx Clock signal on the lower group of pins. This is the default.</td></tr><tr><td>SIO4_ASYNC_CABLE_LEGACY_RXC_UP</td><td>This drives the Rx Clock signal on the lower group of pins.</td></tr></table></td></tr>
<tr><td>cable.
legacy.
rxd_dcd</td><td>This field specifies the legacy configuration of the cable's Rx Data and DCD signals. Valid values are given in the table below. The feature's low-level function is sio4_async_t_cable_legacy_rxd_dcd(). Please also read Cable Configuration Modes</td></tr>
</table>

	(section 5.5, page 42). **Value** / **Description** SIO4_ASYNC_CABLE_LEGACY_RXD_DCD_DISABLE — This disables the signals. SIO4_ASYNC_CABLE_LEGACY_RXD_DCD_LOW — This uses the signals as inputs from the lower group of pins. This is the default. SIO4_ASYNC_CABLE_LEGACY_RXD_DCD_UP — This uses the signals as inputs from the upper group of pins.

4.3.1.3. sio4_async_t.tx

This section describes the structure's transmitter configuration fields.

Field	Description
tx	This structure configures the transmitter portion of the channel.
tx. mode	This field specifies the transmitter's operating mode. Valid values are given in the table below. The feature's low-level function is sio4_async_t_tx_mode(). **Value** / **Description** SIO4_ASYNC_TX_MODE_ASYNC — This selects the Asynchronous operating mode. This is the default and the only valid option for this library.
tx. enable	This field specifies if the transmitter is to be enabled. When configuration is begun (see sio4_async_set(), section 4.1.8, page 14) the transmitter is initialized and disabled. The option in this field is applied towards the end of the configuration process. Valid values are given in the table below. The feature's low-level function is sio4_async_t_tx_enable(). **Value** / **Description** SIO4_ASYNC_TX_ENABLE_NO_AFTER — This disables the transmitter after it has finished the transmission in progress. SIO4_ASYNC_TX_ENABLE_NO_NOW — This disables the transmitter immediately. SIO4_ASYNC_TX_ENABLE_YES_NOW — This enables the transmitter immediately. This is the default. SIO4_ASYNC_TX_ENABLE_YES_W_AE — This enables the transmitter according to the state of any hardware flow control lines.
tx. char_len	This field specifies if the size of transmitted characters. The length specified does not include the Parity Bit, if Parity is enabled. The data bits are the lower significant bits of the byte. Valid values are given in the table below. The feature's low-level function is sio4_async_t_tx_char_len(). **Value** / **Description** SIO4_ASYNC_TX_CHAR_LEN_1 — Characters are 1-bit in length. SIO4_ASYNC_TX_CHAR_LEN_2 — Characters are 2-bits in length. SIO4_ASYNC_TX_CHAR_LEN_3 — Characters are 3-bits in length. SIO4_ASYNC_TX_CHAR_LEN_4 — Characters are 4-bits in length. SIO4_ASYNC_TX_CHAR_LEN_5 — Characters are 5-bits in length. SIO4_ASYNC_TX_CHAR_LEN_6 — Characters are 6-bits in length. SIO4_ASYNC_TX_CHAR_LEN_7 — Characters are 7-bits in length. SIO4_ASYNC_TX_CHAR_LEN_8 — Characters are 8-bits in length. This is the default.
tx. encoding	This field specifies the encoding of the transmitted data. Valid values are given in the table below. The feature's low-level function is sio4_async_t_tx_encoding(). **Value** / **Description** SIO4_ASYNC_TX_ENCODING_BI_MARK — This refers to Biphase Mark encoding. SIO4_ASYNC_TX_ENCODING_BI_LEVEL — This refers to Biphase Level encoding.

	SIO4_ASYNC_TX_ENCODING_BI_SPACE — This refers to Biphase Space encoding. SIO4_ASYNC_TX_ENCODING_D_BI_LEVEL — This refers to Differential Biphase Level encoding. SIO4_ASYNC_TX_ENCODING_NRZ — This refers to NRZ encoding. This is the default. SIO4_ASYNC_TX_ENCODING_NRZB — This refers to NRZB encoding. SIO4_ASYNC_TX_ENCODING_NRZI_MARK — This refers to NRZI-Mark encoding. SIO4_ASYNC_TX_ENCODING_NRZI_SPACE — This refers to NRZI-Space encoding.
tx. bit_rate	This specifies the desired transmission bit rate. During the sio4_async_init_data() call (section 4.1.4, page 10) this is set to the sio4_async_init_t.tx_bit_rate field value. During the call, the library will automatically determine the USC configuration needed to best produce the requested transmit bit rate. Before returning, this field is set to the bit rate that configuration will produce. The feature's low-level function is sio4_async_t_tx_bit_rate(). The Tx Bit Rate is used when the device is being configured, but the value is not recorded for later retrieval.
tx. idle_cond	This field specifies what appears on the Tx Data cable signal while no data is being transmitted. Valid values are given in the table below. The feature's low-level function is sio4_async_t_tx_idle_cond().

Value	Description
SIO4_ASYNC_TX_IDLE_COND_0	The Tx Data signal outputs a continuous data "0" value.
SIO4_ASYNC_TX_IDLE_COND_0_1	The Tx Data signal is alternately data "0" and data "1" values.
SIO4_ASYNC_TX_IDLE_COND_1	The Tx Data signal outputs a continuous data "1" value.
SIO4_ASYNC_TX_IDLE_COND_DEFAULT	The Tx Data signal is driven with the pattern that is the default for the selected serial protocol. This is the default.
SIO4_ASYNC_TX_IDLE_COND_MARK	The Tx Data signal is held high.
SIO4_ASYNC_TX_IDLE_COND_MARK_SPACE	The Tx Data signal is alternately driven with the high then low.
SIO4_ASYNC_TX_IDLE_COND_SPACE	The Tx Data signal is held low.

tx. parity	This structure configures the transmitter's use of Parity checking.
tx. parity. enable	This field enables or disables the use of Parity. When used, the Parity Bit appears to the immediate left of the most significant data bit. Valid values are given in the table below. The feature's low-level function is sio4_async_t_tx_parity_enable().

Value	Description
SIO4_ASYNC_TX_PARITY_ENABLE_NO	Do <u>not</u> generate a Parity bit. This is the default.
SIO4_ASYNC_TX_PARITY_ENABLE_YES	<u>Do</u> generate a Parity bit.

tx. parity. type	This field specifies the type of Parity to use, when its use is enabled. Valid values are given in the table below. The feature's low-level function is sio4_async_t_tx_parity_type().

Value	Description
SIO4_ASYNC_TX_PARITY_TYPE_EVEN	This specifies Even Parity. This is the default.
SIO4_ASYNC_TX_PARITY_TYPE_ODD	This specifies Odd Parity.
SIO4_ASYNC_TX_PARITY_TYPE_ONE	This specifies One Parity (the parity bit is always set).
SIO4_ASYNC_TX_PARITY_TYPE_ZERO	This specifies Zero Parity (the parity bit is always clear).

<table>
<tr><td>tx.
fifo</td><td>This structure configures the transmitter's FIFO parameters.</td></tr>
<tr><td>tx.
fifo.
size</td><td>This field is filled in by the sio4_async_init_data() call (section 4.1.4, page 10) with the size of the channel's Tx FIFO. This is offered for informational purposes only. The feature's low-level function is sio4_async_t_tx_fifo_size().</td></tr>
<tr><td>tx.
fifo.
ae</td><td>This field specifies the Tx FIFO Almost Empty setting. The Tx FIFO Almost Empty status is asserted (goes low) when the Tx FIFO contains this number of values, or fewer. The valid value range is from zero to 0xFFFF. The default is 0x7. The feature's low-level function is sio4_async_t_tx_fifo_ae().</td></tr>
<tr><td>tx.
fifo.
af</td><td>This field specifies the Tx FIFO Almost Full setting. The Tx FIFO Almost Full status is asserted (goes low) when the Tx FIFO contains this number of free spaces, or fewer. The valid value range is from zero to 0xFFFF. The default is 0x7. The feature's low-level function is sio4_async_t_tx_fifo_af().</td></tr>
<tr><td>tx.
fifo.
empty_cfg</td><td>This field configures the transmitter's reaction to the Tx FIFO becoming empty. Valid values are given in the table below. The feature's low-level function is sio4_async_t_tx_fifo_empty_cfg().
<table><tr><th>Value</th><th>Description</th></tr><tr><td>SIO4_ASYNC_TX_FIFO_EMPTY_CFG_IGNORE</td><td>This specifies that the condition is to be ignored. This is the default.</td></tr><tr><td>SIO4_ASYNC_TX_FIFO_EMPTY_CFG_TX_OFF</td><td>This specifies that the transmitter be disabled when the condition occurs.</td></tr></table></td></tr>
<tr><td>tx.
fifo.
space_cfg</td><td>This field configures the FIFO space allocation between the transmitter and the receiver when the Tx FIFO and Rx FIFO are of different sizes. Valid values are given in the table below. The feature's low-level function is sio4_async_t_tx_fifo_space_cfg().
<table><tr><th>Value</th><th>Description</th></tr><tr><td>SIO4_ASYNC_TX_FIFO_SPACE_CFG_RX_2X</td><td>This specifies that the Rx FIFO be twice as large as the Tx FIFO. This is the default.</td></tr><tr><td>SIO4_ASYNC_TX_FIFO_SPACE_CFG_TX_2X</td><td>This specifies that the Tx FIFO be twice as large as the Rx FIFO.</td></tr></table></td></tr>
<tr><td>tx.
io</td><td>This structure configures the transmitter's software settings. These settings are used during sio4_async_write() calls (see section 4.1.10, page 15).</td></tr>
<tr><td>tx.
io.
dma_thresh</td><td>This field configures the minimum size of Block Mode DMA transfers to be performed by the write service. If the Tx FIFO has less than the specified amount of available space, then the driver will wait a single system timer interval before trying again. However, if the Tx FIFO can accommodate what remains of the current request, or if the output stream appears to be idle, then the driver will perform a transfer rather than wait for more space. The valid range is any non-negative value up to the size of the Tx FIFO. The default is 0. The feature's low-level function is sio4_async_t_tx_io_dma_thresh(). Refer to the "Operating Information" section of the SIO4 Reference Manual for additional information.</td></tr>
<tr><td>tx.
io.
mode</td><td>This field selects the mechanism used to transfer data from host memory to the channel's Tx FIFO. Valid values are given in the table below. The feature's low-level function is sio4_async_t_tx_io_mode().
<table><tr><th>Value</th><th>Description</th></tr><tr><td>SIO4_ASYNC_TX_IO_MODE_BMDMA</td><td>This selects Block Mode DMA transfers. *</td></tr><tr><td>SIO4_ASYNC_TX_IO_MODE_DMDMA</td><td>This selects Demand Mode DMA transfers. *</td></tr><tr><td>SIO4_ASYNC_TX_IO_MODE_PIO</td><td>This selects PIO mode transfers. This is the default.</td></tr></table>* The SIO4 has only two DMA engines. A BMDMA or DMDMA transfer request will fail if both DMA engines are already in use by other SIO4 channels.</td></tr>
</table>

tx. io. pio_thresh	This field specifies the threshold for write request sizes that force the use of PIO mode. If a write request is this size or smaller, then the transfer will automatically use PIO. The valid range is any non-negative value. The default is 44. The feature's low-level function is sio4_async_t_tx_io_pio_thresh().
tx. io. timeout	This field specifies the maximum duration of write requests to the driver. The valid range is from zero to 3600. The units are seconds. The default is 10 seconds. The value zero tells the driver to write as much data as possible to the Tx FIFO, but not to wait for additional free space when none is available. The feature's low-level function is sio4_async_t_tx_io_timeout().
tx. io. overrun	This field tells the driver if it is to check for Tx FIFO overrun conditions before proceeding with write requests. Valid values are given in the table below. The feature's low-level function is sio4_async_t_tx_io_overrun().

Value	Description
SIO4_ASYNC_TX_IO_OVERRUN_CHECK	This specifies that the driver should check for overrun conditions. This is the default.
SIO4_ASYNC_TX_IO_OVERRUN_IGNORE	This specifies that the driver should not check for overrun conditions.

4.3.1.4. sio4_async_t.rx

This section describes the structure's receiver configuration fields.

Field	Description
rx	This structure configures the receiver portion of the channel.
rx. mode	This field specifies the receiver's operating mode. Valid values are given in the table below. The feature's low-level function is sio4_async_t_rx_mode().

Value	Description
SIO4_ASYNC_RX_MODE_ASYNC	This selects the Asynchronous operating mode. This is the default and the only valid option for this library.

rx. enable	This field specifies if the receiver is to be enabled. When configuration is begun (see sio4_async_set(), section 4.1.8, page 14) the receiver is initialized and disabled. The option in this field is applied towards the end of the configuration process. Valid values are given in the table below. The feature's low-level function is sio4_async_t_rx_enable().

Value	Description
SIO4_ASYNC_RX_ENABLE_NO_AFTER	This disables the receiver after it has finished the reception in progress.
SIO4_ASYNC_RX_ENABLE_NO_NOW	This disables the receiver immediately.
SIO4_ASYNC_RX_ENABLE_YES_NOW	This enables the receiver immediately. This is the default.
SIO4_ASYNC_RX_ENABLE_YES_W_AE	This enables the receiver according to the state of any hardware flow control lines.

rx. char_len	This field specifies the size of received characters. The length specified does not include the Parity Bit, if Parity is enabled. The data bits are the lower significant bits of the byte. Valid values are given in the table below. The feature's low-level function is sio4_async_t_rx_char_len().

Value	Description
SIO4_ASYNC_RX_CHAR_LEN_1	Characters are 1-bit in length.
SIO4_ASYNC_RX_CHAR_LEN_2	Characters are 2-bits in length.
SIO4_ASYNC_RX_CHAR_LEN_3	Characters are 3-bits in length.
SIO4_ASYNC_RX_CHAR_LEN_4	Characters are 4-bits in length.
SIO4_ASYNC_RX_CHAR_LEN_5	Characters are 5-bits in length.

	SIO4_ASYNC_RX_CHAR_LEN_6 — Characters are 6-bits in length. SIO4_ASYNC_RX_CHAR_LEN_7 — Characters are 7-bits in length. SIO4_ASYNC_RX_CHAR_LEN_8 — Characters are 8-bits in length. This is the default.
rx. encoding	This field specifies the encoding of the received data. Valid values are given in the table below. The feature's low-level function is sio4_async_t_rx_encoding(). **Value** / **Description** SIO4_ASYNC_RX_ENCODING_BI_MARK — This refers to Biphase Mark encoding. SIO4_ASYNC_RX_ENCODING_BI_LEVEL — This refers to Biphase Level encoding. SIO4_ASYNC_RX_ENCODING_BI_SPACE — This refers to Biphase Space encoding. SIO4_ASYNC_RX_ENCODING_D_BI_LEVEL — This refers to Differential Biphase Level encoding. SIO4_ASYNC_RX_ENCODING_NRZ — This refers to NRZ encoding. This is the default. SIO4_ASYNC_RX_ENCODING_NRZB — This refers to NRZB encoding. SIO4_ASYNC_RX_ENCODING_NRZI_MARK — This refers to NRZI-Mark encoding. SIO4_ASYNC_RX_ENCODING_NRZI_SPACE — This refers to NRZI-Space encoding.
rx. bit_rate	This specifies the desired receive bit rate. During the sio4_async_init_data() call (section 4.1.4, page 10) this is set to the sio4_async_init_t.rx_bit_rate field provided to the call. During the call, the library will automatically determine the USC configuration needed to best produce the requested receive bit rate. Before returning, this field is set to the bit rate that configuration will produce. The feature's low-level function is sio4_async_t_rx_bit_rate(). The Tx Bit Rate is used when the device is being configured, but the value is not recorded for later retrieval.
rx. sync_byte	This specifies the value to be compared to received data as the data enters the Rx FIFO (the one outside the USC). This comparison can be used for interrupt generation. Valid values are from zero to 0xFF. The default is zero. The feature's low-level function is sio4_async_t_rx_sync_byte().
rx. status_word	This field controls whether the firmware will place the USC Receive Control/Status Register in the Rx FIFO along with the received data. Valid values are given in the table below. The feature's low-level function is sio4_async_t_rx_status_word(). **Value** / **Description** SIO4_ASYNC_RX_STATUS_WORD_DISABLE — The RCSR is not placed in the Rx FIFO. This is the default. SIO4_ASYNC_RX_STATUS_WORD_ENABLE — The RCSR is placed in the Rx FIFO.
rx. parity	This structure configures the receiver's use of Parity checking.
rx. parity. enable	This field enables or disables the use of Parity. When enabled, the character size does not include the Parity Bit. When used, the Parity Bit appears to the immediate left of the most significant data bit. Valid values are given in the table below. The feature's low-level function is sio4_async_t_rx_parity_enable(). **Value** / **Description** SIO4_ASYNC_RX_PARITY_ENABLE_NO — Do not generate a Parity bit. This is the default. SIO4_ASYNC_RX_PARITY_ENABLE_YES — Do generate a Parity bit.
rx. parity. type	This field specifies the type of Parity to use, when its use is enabled. Valid values are given in the table below. The feature's low-level function is sio4_async_t_rx_parity_type(). **Value** / **Description** SIO4_ASYNC_RX_PARITY_TYPE_EVEN — This specifies Even Parity. This is the default. SIO4_ASYNC_RX_PARITY_TYPE_ODD — This specifies Odd Parity.

<table>
<tr><td></td><td>
<table>
<tr><td>SIO4_ASYNC_RX_PARITY_TYPE_ONE</td><td>This specifies One Parity (the parity bit is always set).</td></tr>
<tr><td>SIO4_ASYNC_RX_PARITY_TYPE_ZERO</td><td>This specifies Zero Parity (the parity bit is always clear).</td></tr>
</table>
</td></tr>
<tr><td>rx.
fifo</td><td>This structure configures the receiver's FIFO parameters.</td></tr>
<tr><td>rx.
fifo.
size</td><td>This field is filled in by the sio4_async_init_data() call (section 4.1.4, page 10) with the size of the channel's Rx FIFO. This is offered for informational purposes only. The feature's low-level function is sio4_async_t_rx_fifo_size().</td></tr>
<tr><td>rx.
fifo.
ae</td><td>This field specifies the Rx FIFO Almost Empty setting. The Rx FIFO Almost Empty status is asserted (goes low) when the Rx FIFO contain this number of values, or fewer. The valid value range is from zero to 0xFFFF. The default is 0x7. The feature's low-level function is sio4_async_t_rx_fifo_ae().</td></tr>
<tr><td>rx.
fifo.
af</td><td>This field specifies the Rx FIFO Almost Full setting. The Rx FIFO Almost Full status is asserted (goes low) when the Rx FIFO contain this number of values, or fewer. The valid value range is from zero to 0xFFFF. The default is 0x7. The feature's low-level function is sio4_async_t_rx_fifo_af().</td></tr>
<tr><td>rx.
fifo.
full_cfg</td><td>This field configures the receiver's reaction to the Rx FIFO becoming full. Valid values are given in the table below. The feature's low-level function is sio4_async_t_rx_fifo_full_cfg(). This field refers to the channel specific setting, when supported. The corresponding global setting is not supported by the Asynchronous Protocol Library because it affects channels other than the one being accessed. The global setting must be handled separately by the application.
<table>
<tr><th>Value</th><th>Description</th></tr>
<tr><td>SIO4_ASYNC_RX_FIFO_FULL_CFG_DISABLE</td><td>This specifies that the receiver be disabled when the condition occurs.</td></tr>
<tr><td>SIO4_ASYNC_RX_FIFO_FULL_CFG_OVER</td><td>This specifies that the condition produce an overflow. This is the default.</td></tr>
</table>
</td></tr>
<tr><td>rx.
io</td><td>This structure configures the receiver's software settings. These settings are used during sio4_async_read() calls (section 4.1.6.1, page 13).</td></tr>
<tr><td>rx.
io.
dma_thresh</td><td>This field specifies the minimum size of Block Mode DMA transfers when the driver needs to wait for additional data to become available in the Rx FIFO. If data is available in the Rx FIFO, but it is less than the specified threshold and won't fulfill the request, then the driver will wait for additional data become available. The wait period is one system timer tick. The valid range is any non-negative value up to the size of the Rx FIFO. The default is 0. The feature's low-level function is sio4_async_t_rx_io_dma_thresh(). Refer to the "Operating Information" section of the SIO4 Reference Manual for additional information.</td></tr>
<tr><td>rx.
io.
mode</td><td>This field selects the mechanism used to transfer data from the channel's Rx FIFO to host memory. Valid values are given in the table below. The feature's low-level function is sio4_async_t_rx_io_mode().
<table>
<tr><th>Value</th><th>Description</th></tr>
<tr><td>SIO4_ASYNC_RX_IO_MODE_BMDMA</td><td>This selects Block Mode DMA transfers. *</td></tr>
<tr><td>SIO4_ASYNC_RX_IO_MODE_DMDMA</td><td>This selects Demand Mode DMA mode transfers. *</td></tr>
<tr><td>SIO4_ASYNC_RX_IO_MODE_PIO</td><td>This selects PIO mode transfers. This is the default.</td></tr>
</table>
* The SIO4 has only two DMA engines. A BMDMA or DMDMA transfer request will fail if both DMA engines are already in use by other SIO4 channels.</td></tr>
<tr><td>rx.
io.
pio_thresh</td><td>This field specifies the threshold for read request sizes that force the use of PIO mode. If a read request is this size or smaller, then the transfer will automatically use PIO. The valid range is any non-negative value. The default is 44. The feature's low-level function is sio4_async_t_rx_io_pio_thresh().</td></tr>
</table>

<table>
<tr><td>rx.
io.
timeout</td><td>This field specifies the maximum duration of read requests to the driver. The valid range is from zero to 3600. The units are seconds. The default is 10 seconds. The value zero tells the driver to read as much data as possible from the Rx FIFO, but not to wait for additional data when none is available. The feature's low-level function is sio4_async_t_rx_io_timeout().</td></tr>
<tr><td>rx.
io.
overrun</td><td>This field tells the driver if it is to check for Rx FIFO overrun conditions before proceeding with read requests. Valid values are given in the table below. The feature's low-level function is sio4_async_t_rx_io_overrun().
<table><tr><th>Value</th><th>Description</th></tr><tr><td>SIO4_ASYNC_RX_IO_OVERRUN_CHECK</td><td>This specifies that the driver check for overrun conditions. This is the default.</td></tr><tr><td>SIO4_ASYNC_RX_IO_OVERRUN_IGNORE</td><td>This specifies that the driver not check for overrun conditions.</td></tr></table></td></tr>
<tr><td>rx.
io.
underrun</td><td>This field tells the driver if it is to check for Rx FIFO underrun conditions before proceeding with read requests. Valid values are given in the table below. The feature's low-level function is sio4_async_t_rx_io_underrun().
<table><tr><th>Value</th><th>Description</th></tr><tr><td>SIO4_ASYNC_RX_IO_UNDERRUN_CHECK</td><td>This specifies that the driver check for underrun conditions. This is the default.</td></tr><tr><td>SIO4_ASYNC_RX_IO_UNDERRUN_IGNORE</td><td>This specifies that the driver not check for underrun conditions.</td></tr></table></td></tr>
<tr><td>rx.
time_stamp</td><td>This structure configures the receiver's Time Stamp settings.</td></tr>
<tr><td>rx.
time_stamp.
enable</td><td>This field enables or disables the channel's use of the Time Stamp feature. Valid values are given in the table below. The feature's low-level function is sio4_async_t_rx_time_stamp_enable().
<table><tr><th>Value</th><th>Description</th></tr><tr><td>SIO4_ASYNC_RX_TIME_STAMP_ENABLE_NO</td><td>Do not use the Time Stamp feature. This is the default.</td></tr><tr><td>SIO4_ASYNC_RX_TIME_STAMP_ENABLE_YES</td><td>Do use the Time Stamp feature.</td></tr></table></td></tr>
<tr><td>rx.
time_stamp.
clk_src</td><td>This field selects the Time Stamp clock source. Valid values are given in the table below. The feature's low-level function is sio4_async_t_rx_time_stamp_clk_src().
<table><tr><th>Value</th><th>Description</th></tr><tr><td>SIO4_ASYNC_RX_TIME_STAMP_CLK_SRC_EXT</td><td>Use the board's external TTL clock source. *</td></tr><tr><td>SIO4_ASYNC_RX_TIME_STAMP_CLK_SRC_INT</td><td>Use the board's internal 1us clock. This is the default. *</td></tr></table>* All four channels on the SIO4 use the same clock source.</td></tr>
</table>

4.3.1.5. sio4_async_t.usc

This section describes the structure's USC configuration fields.

<table>
<tr><th>Field</th><th>Description</th></tr>
<tr><td>usc</td><td>This structure configures the remaining USC portion of the channel.</td></tr>
<tr><td>usc.
mode</td><td>This field specifies the USC's overall operating mode. Valid values are given in the table below. The feature's low-level function is sio4_async_t_usc_mode().
<table><tr><th>Value</th><th>Description</th></tr><tr><td>SIO4_ASYNC_USC_MODE_AUTO_ECHO</td><td>This is the USC's Auto Echo mode.</td></tr><tr><td>SIO4_ASYNC_USC_MODE_LOOPBACK_EXT</td><td>This is the USC's external loopback mode.</td></tr></table></td></tr>
</table>

	SIO4_ASYNC_USC_MODE_LOOPBACK_INT — This is the USC's internal loopback mode. SIO4_ASYNC_USC_MODE_NORMAL — This is the USC's normal operating mode. This is the default.
usc. txd	This field configures the operation of the USC's Tx Data pin. Valid values are given in the table below. The feature's low-level function is `sio4_async_t_usc_txd()`. **Value** / **Description** SIO4_ASYNC_USC_TXD_OUT_0 — The pin is driven low. SIO4_ASYNC_USC_TXD_OUT_1 — The pin is driven high. SIO4_ASYNC_USC_TXD_OUT_TXD — The pin is driven from the transmitter's Tx Data signal. This is the default. SIO4_ASYNC_USC_TXD_TRI — The pin is tri-stated.
usc. cts	This field configures the operation of the USC's CTS pin. Valid values are given in the table below. The feature's low-level function is `sio4_async_t_usc_cts()`. **Value** / **Description** SIO4_ASYNC_USC_CTS_OUT_0 — The pin is driven low. SIO4_ASYNC_USC_CTS_OUT_1 — The pin is driven high. SIO4_ASYNC_USC_CTS_IN_CBL_CTS — The pin is an input driver from the cable's CTS signal. SIO4_ASYNC_USC_CTS_TRI — The pin is tri-stated. This is the default.
usc. cts_legacy	This field configures the operation of the USC's CTS pin for legacy mode cable interface configurations. Valid values are given in the table below. The feature's low-level function is `sio4_async_t_usc_cts_legacy()`. Please also read Cable Configuration Modes (section 5.5, page 42). **Value** / **Description** SIO4_ASYNC_USC_TX_CTS_LEG_IN — The pin operates as an input. This is the default. SIO4_ASYNC_USC_TX_CTS_LEG_OUT_0 — The pin operates as an output driven low. SIO4_ASYNC_USC_TX_CTS_LEG_OUT_1 — The pin operates as an output driven high.
usc. dcd	This field configures the operation of the USC's DCD pin. Valid values are given in the table below. The feature's low-level function is `sio4_async_t_usc_dcd()`. **Value** / **Description** SIO4_ASYNC_USC_DCD_DISABLE — The pin is disabled. This is the default. SIO4_ASYNC_USC_DCD_IN_DCD_CBL_DCD — The pin is an input for the receiver's DCD function and is driven from the cable's DCD signal. SIO4_ASYNC_USC_DCD_IN_SYNC_CBL_DCD — The pin is an input for the receiver's SYNC function and is driven from the cable's DCD signal. SIO4_ASYNC_USC_DCD_OUT_0 — The pin is driven low. * SIO4_ASYNC_USC_DCD_OUT_1 — The pin is driven high. * * This option enables the cable DCD signal to be driven, though the `cable.dcd` field (section 4.3.1.2, page 20) may configure the cable to output an alternate signal.
usc. dcd_legacy	This field configures the operation of the USC's DCD pin for legacy mode cable interface configurations. Valid values are given in the table below. The feature's low-level function is `sio4_async_t_usc_dcd_legacy()`. Please also read Cable Configuration Modes (section 5.5, page 42). **Value** / **Description** SIO4_ASYNC_USC_DCD_LEG_IN_DCD — The pin operates as a DCD input. This is the default. SIO4_ASYNC_USC_DCD_LEG_IN_SYNC — The pin operates as a SYNC input.

	SIO4_ASYNC_USC_DCD_LEG_OUT_0 — The pin operates as an output driven low. SIO4_ASYNC_USC_DCD_LEG_OUT_1 — The pin operates as an output driven high.
usc. tx	This structure configures the remaining USC transmitter settings.
usc. tx. clk_rate	This field configures the clock divider rate for the USC transmitter. This corresponds to the oversampling rate used by the receiver. Valid values are given in the table below. The feature's low-level function is sio4_async_t_usc_tx_clk_rate(). **Value** — **Description** SIO4_ASYNC_USC_TX_CLK_RATE_16X — Divide the source by 16. SIO4_ASYNC_USC_TX_CLK_RATE_32X — Divide the source by 32. SIO4_ASYNC_USC_TX_CLK_RATE_64X — Divide the source by 64. * * This is the hard coded initialization default, which may subsequently be modified as needed to produce the user specified Tx bit rate.
usc. tx. clk_src	This field configures the source for the USC transmitter clock. Valid values are given in the table below. The feature's low-level function is sio4_async_t_usc_tx_clk_src(). **Value** — **Description** SIO4_ASYNC_USC_TX_CLK_SRC_BRG0 — Select Baud Rate Generator 0. SIO4_ASYNC_USC_TX_CLK_SRC_BRG1 — Select Baud Rate Generator 1. SIO4_ASYNC_USC_TX_CLK_SRC_CTR0 — Select Counter 0. SIO4_ASYNC_USC_TX_CLK_SRC_CTR1 — Select Counter 1. SIO4_ASYNC_USC_TX_CLK_SRC_DISABLE — Disable the transmitter. SIO4_ASYNC_USC_TX_CLK_SRC_DPLL — Select the DPLL. SIO4_ASYNC_USC_TX_CLK_SRC_RXC_PIN — Select the Rx Clock pin. * SIO4_ASYNC_USC_TX_CLK_SRC_TXC_PIN — Select the Tx Clock pin. * This is the hard coded initialization default, which may subsequently be modified as needed to produce the user specified Tx bit rate.
usc. tx. txc	This field configures the operation of the USC's Tx Clock pin. Valid values are given in the table below. The feature's low-level function is sio4_async_t_usc_tx_txc(). **Value** — **Description** SIO4_ASYNC_USC_TX_TXC_IN_0 — The pin is an input driven low. SIO4_ASYNC_USC_TX_TXC_IN_1 — The pin is an input driven high. SIO4_ASYNC_USC_TX_TXC_IN_CBL_RXAUX — The pin is an input driven from the cable's Rx Aux signal. SIO4_ASYNC_USC_TX_TXC_IN_CBL_RXC — The pin is an input driven from the cable's Rx Clock signal. SIO4_ASYNC_USC_TX_TXC_IN_OSC — The pin is an input driven from the onboard oscillator. SIO4_ASYNC_USC_TX_TXC_IN_OSC_INV — The pin is an input driven from the inverted onboard oscillator. SIO4_ASYNC_USC_TX_TXC_OUT_BRG0 — The pin is an output driven from Baud Rate Generator 0. SIO4_ASYNC_USC_TX_TXC_OUT_BRG1 — The pin is an output driven from Baud Rate Generator 1. SIO4_ASYNC_USC_TX_TXC_OUT_BYTE_CLK — The pin is an output driven from the transmitter's Byte Clock. SIO4_ASYNC_USC_TX_TXC_OUT_CLK — The pin is an output driven from the transmit clock. This is the default. SIO4_ASYNC_USC_TX_TXC_OUT_COMP — The pin is an output driven from the transmit complete signal. SIO4_ASYNC_USC_TX_TXC_OUT_CTR1 — The pin is an output driven from Counter 1.

	SIO4_ASYNC_USC_TX_TXC_OUT_DPLL_TX — The pin is an output driven from the transmit clock from the DPLL.
usc. tx. txc_legacy	This field configures the operation of the USC's Tx Clock pin for legacy mode cable interface configurations. Valid values are given in the table below. The feature's low-level function is `sio4_async_t_usc_tx_txc_legacy()`. Please also read Cable Configuration Modes (section 5.5, page 42).

Value	Description
SIO4_ASYNC_USC_TX_TXC_LEG_IN	The pin operates as an input.
SIO4_ASYNC_USC_TX_TXC_LEG_OUT_BRG0	The pin is an output driven from Baud Rate Generator 0.
SIO4_ASYNC_USC_TX_TXC_LEG_OUT_BRG1	The pin is an output driven from Baud Rate Generator 1.
SIO4_ASYNC_USC_TX_TXC_LEG_OUT_BYTE_CLK	The pin is an output driven from the transmitter's Byte Clock.
SIO4_ASYNC_USC_TX_TXC_LEG_OUT_CLK	The pin is an output driven from the transmit clock. This is the default.
SIO4_ASYNC_USC_TX_TXC_LEG_OUT_COMP	The pin is an output driven from the transmit complete signal.
SIO4_ASYNC_USC_TX_TXC_LEG_OUT_CTR1	The pin is an output driven from Counter 1.
SIO4_ASYNC_USC_TX_TXC_LEG_OUT_DPLL_TX	The pin is an output driven from the transmit clock from the DPLL.

usc.
tx.
stop_bits

This field configures the number of Stop Bits injected by the transmitter after each data byte. Valid values are given in the table below. The feature's low-level function is `sio4_async_t_usc_tx_stop_bits()`.

Value	Description
SIO4_ASYNC_USC_TX_STOP_BIT_1	This refers to a single stop bit. This is the default.
SIO4_ASYNC_USC_TX_STOP_BIT_2	This refers to two stop bits.
SIO4_ASYNC_USC_TX_STOP_BIT_0__9_16	This refers to 9/16th of a stop bit.
SIO4_ASYNC_USC_TX_STOP_BIT_0_10_16	This refers to 10/16th of a stop bit (5/8th).
SIO4_ASYNC_USC_TX_STOP_BIT_0_11_16	This refers to 11/16th of a stop bit.
SIO4_ASYNC_USC_TX_STOP_BIT_0_12_16	This refers to 12/16th of a stop bit (3/4th).
SIO4_ASYNC_USC_TX_STOP_BIT_0_13_16	This refers to 13/16th of a stop bit.
SIO4_ASYNC_USC_TX_STOP_BIT_0_14_16	This refers to 14/16th of a stop bit (7/8th).
SIO4_ASYNC_USC_TX_STOP_BIT_0_15_16	This refers to 15/16th of a stop bit.
SIO4_ASYNC_USC_TX_STOP_BIT_1__1_16	This refers to 1 and 1/16th stop bits.
SIO4_ASYNC_USC_TX_STOP_BIT_1__2_16	This refers to 1 and 2/16th stop bits (1+1/8th).
SIO4_ASYNC_USC_TX_STOP_BIT_1__3_16	This refers to 1 and 3/16th stop bits.
SIO4_ASYNC_USC_TX_STOP_BIT_1__4_16	This refers to 1 and 4/16th stop bits (1+1/4th).
SIO4_ASYNC_USC_TX_STOP_BIT_1__5_16	This refers to 1 and 5/16th stop bits.
SIO4_ASYNC_USC_TX_STOP_BIT_1__6_16	This refers to 1 and 6/16th stop bits (1+3/8th).
SIO4_ASYNC_USC_TX_STOP_BIT_1__7_16	This refers to 1 and 7/16th stop bits.
SIO4_ASYNC_USC_TX_STOP_BIT_1__8_16	This refers to 1 and 8/16th stop bits (1+1/2).
SIO4_ASYNC_USC_TX_STOP_BIT_1__9_16	This refers to 1 and 9/16th stop bits.
SIO4_ASYNC_USC_TX_STOP_BIT_1_10_16	This refers to 1 and 10/16th stop bits (1+5/8th).

	SIO4_ASYNC_USC_TX_STOP_BIT_1_11_16 — This refers to 1 and 11/16th stop bits. SIO4_ASYNC_USC_TX_STOP_BIT_1_12_16 — This refers to 1 and 12/16th stop bits (1+3/4th). SIO4_ASYNC_USC_TX_STOP_BIT_1_13_16 — This refers to 1 and 13/16th stop bits. SIO4_ASYNC_USC_TX_STOP_BIT_1_14_16 — This refers to 1 and 14/16th stop bits (1+7/8th). SIO4_ASYNC_USC_TX_STOP_BIT_1_15_16 — This refers to 1 and 15/16th stop bits.
usc. rx	This structure configures the remaining USC receiver settings.
usc. rx. clk_rate	This field configures the clock divider rate for the USC receiver. This is the receiver oversampling rate. Valid values are given in the table below. The feature's low-level function is sio4_async_t_usc_rx_clk_rate(). **Value** / **Description** SIO4_ASYNC_USC_RX_CLK_RATE_16X — Divide the source by 16. SIO4_ASYNC_USC_RX_CLK_RATE_32X — Divide the source by 32. SIO4_ASYNC_USC_RX_CLK_RATE_64X — Divide the source by 64. * * This is the hard coded initialization default, which may subsequently be modified as needed to produce the user specified Rx bit rate.
usc. rx. clk_src	This field configures the source for the USC receiver clock. Valid values are given in the table below. The feature's low-level function is sio4_async_t_usc_rx_clk_src(). **Value** / **Description** SIO4_ASYNC_USC_RX_CLK_SRC_BRG0 — Select Baud Rate Generator 0. SIO4_ASYNC_USC_RX_CLK_SRC_BRG1 — Select Baud Rate Generator 1. SIO4_ASYNC_USC_RX_CLK_SRC_CTR0 — Select Counter 0. SIO4_ASYNC_USC_RX_CLK_SRC_CTR1 — Select Counter 1. SIO4_ASYNC_USC_RX_CLK_SRC_DISABLE — Disable the receiver. SIO4_ASYNC_USC_RX_CLK_SRC_DPLL — Select the DPLL. SIO4_ASYNC_USC_RX_CLK_SRC_RXC_PIN — Select the Rx Clock pin. * SIO4_ASYNC_USC_RX_CLK_SRC_TXC_PIN — Select the Tx Clock pin. * This is the hard coded initialization default, which may subsequently be modified as needed to produce the user specified Rx bit rate.
usc. rx. rxc	This field configures the operation of the USC's Rx Clock pin. Valid values are given in the table below. The feature's low-level function is sio4_async_t_usc_rx_rxc(). **Value** / **Description** SIO4_ASYNC_USC_RX_RXC_IN_0 — The pin is an input driven low. SIO4_ASYNC_USC_RX_RXC_IN_1 — The pin is an input driven high. SIO4_ASYNC_USC_RX_RXC_IN_CBL_RXAUX — The pin is an input driven from the cable's Rx Aux signal. SIO4_ASYNC_USC_RX_RXC_IN_CBL_RXC — The pin is an input driven from the cable's Rx Clock signal. This is the default. SIO4_ASYNC_USC_RX_RXC_IN_OSC — The pin is an input driven from the onboard oscillator. SIO4_ASYNC_USC_RX_RXC_IN_OSC_INV — The pin is an input driven from the inverted onboard oscillator. SIO4_ASYNC_USC_RX_RXC_OUT_BRG0 — The pin is an output driven from Baud Rate Generator 0. SIO4_ASYNC_USC_RX_RXC_OUT_BRG1 — The pin is an output driven from Baud Rate Generator 1. SIO4_ASYNC_USC_RX_RXC_OUT_BYTE_CLK — The pin is an output driven from the receiver's Byte Clock.

	SIO4_ASYNC_USC_RX_RXC_OUT_CLK — The pin is an output driven from the receiver clock. SIO4_ASYNC_USC_RX_RXC_OUT_CTR0 — The pin is an output driven from Counter 0. SIO4_ASYNC_USC_RX_RXC_OUT_DPLL_RX — The pin is an output driven from the DPLL receiver clock. SIO4_ASYNC_USC_RX_RXC_OUT_SYNC — The pin is an output driven from the receiver's SYNC signal.
usc. rx. rxc_legacy	This field configures the operation of the USC's Rx Clock pin for legacy mode cable interface configurations. Valid values are given in the table below. The feature's low-level function is sio4_async_t_usc_rx_rxc_legacy(). Please also read Cable Configuration Modes (section 5.5, page 42). **Value** — **Description** SIO4_ASYNC_USC_RX_RXC_LEG_IN — The pin is an input. This is the default. SIO4_ASYNC_USC_RX_RXC_LEG_OUT_BRG0 — The pin is an output driven from Baud Rate Generator 0. SIO4_ASYNC_USC_RX_RXC_LEG_OUT_BRG1 — The pin is an output driven from Baud Rate Generator 1. SIO4_ASYNC_USC_RX_RXC_LEG_OUT_BYTE_CLK — The pin is an output driven from the receiver's Byte Clock. SIO4_ASYNC_USC_RX_RXC_LEG_OUT_CLK — The pin is an output driven from the receiver clock. SIO4_ASYNC_USC_RX_RXC_LEG_OUT_CTR0 — The pin is an output driven from Counter 0. SIO4_ASYNC_USC_RX_RXC_LEG_OUT_DPLL_RX — The pin is an output driven from the DPLL receiver clock. SIO4_ASYNC_USC_RX_RXC_LEG_OUT_SYNC — The pin is an output driven from the receiver's SYNC signal.
usc. brg0	This structure configures settings for Baud Rate Generator 0 (BRG0).
usc. brg0. enable	This field enables or disabled Baud Rate Generator 0. Valid values are given in the table below. The feature's low-level function is sio4_async_t_usc_brg0_enable(). **Value** — **Description** SIO4_ASYNC_USC_BRG0_ENABLE_NO — This disables BRG0. * SIO4_ASYNC_USC_BRG0_ENABLE_YES — This enables BRG0. * This is the hard coded initialization default, which may subsequently be modified as needed to produce the user specified Tx bit rate.
usc. brg0. clk_src	This field selects the clock source for Baud Rate Generator 0. Valid values are given in the table below. The feature's low-level function is sio4_async_t_usc_brg0_clk_src(). **Value** — **Description** SIO4_ASYNC_USC_BRG0_CLK_SRC_CTR0 — This selects the Counter 0 output. * SIO4_ASYNC_USC_BRG0_CLK_SRC_CTR1 — This selects the Counter 1 output. SIO4_ASYNC_USC_BRG0_CLK_SRC_RXC_PIN — This selects the signal present at the USC's Rx Clock pin. SIO4_ASYNC_USC_BRG0_CLK_SRC_TXC_PIN — This selects the signal present at the USC's Tx Clock pin. * This is the hard coded initialization default, which may subsequently be modified as needed to produce the user specified Tx bit rate.

<table>
<tr><td>usc.
brg0.
divider</td><td>This field specifies the clock divider value for Baud Rate Generator 0. The valid value range is from zero to 0xFFFF. The hard coded initialization default is zero, which may subsequently be modified as needed to produce the user specified Tx bit rate. The feature's low-level function is sio4_async_t_usc_brg0_divider().</td></tr>
<tr><td>usc.
brg0.
mode</td><td>This field specifies the Baud Rate Generator 0 operating mode. Valid values are given in the table below. The feature's low-level function is sio4_async_t_usc_brg0_mode().
<table><tr><th>Value</th><th>Description</th></tr><tr><td>SIO4_ASYNC_USC_BRG0_MODE_CONT</td><td>This selects continuous operation. This is the default.</td></tr><tr><td>SIO4_ASYNC_USC_BRG0_MODE_SINGLE</td><td>This selects single shot mode, in which clocking stops when the counter value reaches zero.</td></tr></table></td></tr>
<tr><td>usc.
brg1</td><td>This structure configures settings for Baud Rate Generator 1 (BRG1).</td></tr>
<tr><td>usc.
brg1.
enable</td><td>This field enables or disabled Baud Rate Generator 1. Valid values are given in the table below. The feature's low-level function is sio4_async_t_usc_brg1_enable().
<table><tr><th>Value</th><th>Description</th></tr><tr><td>SIO4_ASYNC_USC_BRG1_ENABLE_NO</td><td>This disables BRG1. *</td></tr><tr><td>SIO4_ASYNC_USC_BRG1_ENABLE_YES</td><td>This enables BRG1.</td></tr></table>* This is the hard coded initialization default, which may subsequently be modified as needed to produce the user specified Rx bit rate.</td></tr>
<tr><td>usc.
brg1.
clk_src</td><td>This field selects the clock source for Baud Rate Generator 1. Valid values are given in the table below. The feature's low-level function is sio4_async_t_usc_brg1_clk_src().
<table><tr><th>Value</th><th>Description</th></tr><tr><td>SIO4_ASYNC_USC_BRG1_CLK_SRC_CTR0</td><td>This selects the output from Counter 0.</td></tr><tr><td>SIO4_ASYNC_USC_BRG1_CLK_SRC_CTR1</td><td>This selects the output from Counter 1. This is the default.</td></tr><tr><td>SIO4_ASYNC_USC_BRG1_CLK_SRC_RXC_PIN</td><td>This selects the signal present at the USC's Rx Clock pin.</td></tr><tr><td>SIO4_ASYNC_USC_BRG1_CLK_SRC_TXC_PIN</td><td>This selects the signal present at the USC's Tx Clock pin.</td></tr></table></td></tr>
<tr><td>usc.
brg1.
divider</td><td>This field specifies the clock divider value for Baud Rate Generator 1. The valid value range is from zero to 0xFFFF. The hard coded initialization default is zero, which may subsequently be modified as needed to produce the user specified Rx bit rate. The feature's low-level function is sio4_async_t_usc_brg1_divider().</td></tr>
<tr><td>usc.
brg1.
mode</td><td>This field specifies the Baud Rate Generator 1 operating mode. Valid values are given in the table below. The feature's low-level function is sio4_async_t_usc_brg1_mode().
<table><tr><th>Value</th><th>Description</th></tr><tr><td>SIO4_ASYNC_USC_BRG1_MODE_CONT</td><td>This selects continuous operation. This is the default.</td></tr><tr><td>SIO4_ASYNC_USC_BRG1_MODE_SINGLE</td><td>This selects single shot mode, in which clocking stops when the counter value reaches zero.</td></tr></table></td></tr>
<tr><td>usc.
ctr0</td><td>This structure configures settings for Counter 0 (CTR0).</td></tr>
<tr><td>usc.
ctr0.
clk_src</td><td>This field selects the clock source for Counter 0. Valid values are given in the table below. The feature's low-level function is sio4_async_t_usc_ctr0_clk_src().
<table><tr><th>Value</th><th>Description</th></tr><tr><td>SIO4_ASYNC_USC_CTR0_CLK_SRC_DISABLE</td><td>This disables Counter 0. *</td></tr><tr><td>SIO4_ASYNC_USC_CTR0_CLK_SRC_RXC_PIN</td><td>This selects the signal present at the</td></tr></table></td></tr>
</table>

	<table><tr><td></td><td>USC's Rx Clock pin.</td></tr><tr><td>SIO4_ASYNC_USC_CTR0_CLK_SRC_TXC_PIN</td><td>This selects the signal present at the USC's Tx Clock pin.</td></tr></table> * This is the hard coded initialization default, which may subsequently be modified as needed to produce the user specified Tx bit rate.
usc. ctr0. rate	This field selects the divider rate for Counter 0. Valid values are given in the table below. The feature's low-level function is sio4_async_t_usc_ctr0_rate(). <table><tr><th>Value</th><th>Description</th></tr><tr><td>SIO4_ASYNC_USC_CTR0_RATE_4X</td><td>This sets the output as the input divided by four.</td></tr><tr><td>SIO4_ASYNC_USC_CTR0_RATE_8X</td><td>This sets the output as the input divided by eight.</td></tr><tr><td>SIO4_ASYNC_USC_CTR0_RATE_16X</td><td>This sets the output as the input divided by 16.</td></tr><tr><td>SIO4_ASYNC_USC_CTR0_RATE_32X</td><td>This sets the output as the input divided by 32. *</td></tr></table> * This is the hard coded initialization default, which may subsequently be modified as needed to produce the user specified Tx bit rate.
usc. ctr1	This structure configures settings for Counter 1 (CTR1).
usc. ctr1. clk_src	This field selects the clock source for Counter 1. Valid values are given in the table below. The feature's low-level function is sio4_async_t_usc_ctr1_clk_src(). <table><tr><th>Value</th><th>Description</th></tr><tr><td>SIO4_ASYNC_USC_CTR1_CLK_SRC_DISABLE</td><td>This disables Counter 1. *</td></tr><tr><td>SIO4_ASYNC_USC_CTR1_CLK_SRC_RXC_PIN</td><td>This selects the signal present at the USC's Rx Clock pin.</td></tr><tr><td>SIO4_ASYNC_USC_CTR1_CLK_SRC_TXC_PIN</td><td>This selects the signal present at the USC's Tx Clock pin.</td></tr></table> * This is the hard coded initialization default, which may subsequently be modified as needed to produce the user specified Rx bit rate.
usc. ctr1. rate_src	This field selects the source for the rate divider used by Counter 1. Valid values are given in the table below. The feature's low-level function is sio4_async_t_usc_ctr1_rate_src(). <table><tr><th>Value</th><th>Description</th></tr><tr><td>SIO4_ASYNC_USC_CTR1_RATE_SRC_CTR0</td><td>This selects the rate divider used by CTR0.</td></tr><tr><td>SIO4_ASYNC_USC_CTR1_RATE_SRC_DPLL</td><td>This selects the rate divider used by the DPLL. This is the default.</td></tr></table>
usc. dpll	This structure configures settings for DPLL. The DPLL is unused by the Asynchronous Protocol Library except for the one field described below.
usc. dpll. rate	This field selects the divider rate for the DPLL, which is used as the divider rate for Counter 1 (CTR1). Valid values are given in the table below. The feature's low-level function is sio4_async_t_usc_dpll_rate(). <table><tr><th>Value</th><th>Description</th></tr><tr><td>SIO4_ASYNC_USC_DPLL_RATE_CTR1_4X</td><td>This sets the output as the input divided by four. This option is usable only when the DPLL divider rate is being used as the divider rate for CTR1.</td></tr><tr><td>SIO4_ASYNC_USC_DPLL_RATE_8X</td><td>This sets the output as the input divided by eight.</td></tr><tr><td>SIO4_ASYNC_USC_DPLL_RATE_16X</td><td>This sets the output as the input divided by 16.</td></tr><tr><td>SIO4_ASYNC_USC_DPLL_RATE_32X</td><td>This sets the output as the input divided by 32. *</td></tr></table> * This is the hard coded initialization default, which may subsequently be modified as needed to produce the user specified Rx bit rate.

5. Operating Information

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

5.1. Basic Illustration

The below figure is included to assist individuals in the configuration of the SIO4. The figure illustrates boards with more recent firmware. The DMA references are handled automatically by the driver to facilitate movement of data between the USC and the on-board FIFOs.

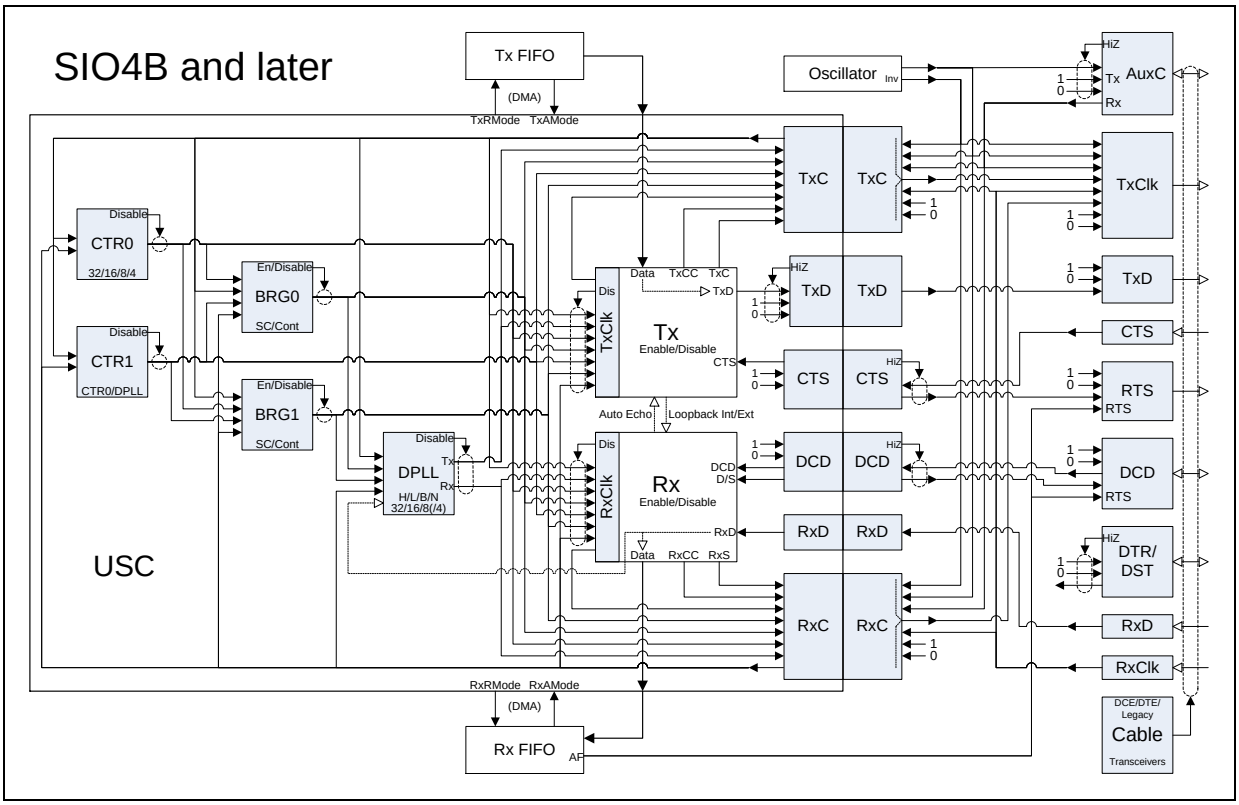

Figure 2 A functional illustration of an SIO4B or later model board.

5.2. Getting Started

To configure the SIO4 for Asynchronous operation meeting specific requirements, the recommended starting point is a local, customizable copy of the `asyncc2c` sample application included with the driver. This application is designed to transfer bulk data between an SIO4 transmitter and an SIO4 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 totally onboard the SIO4 without driving the cable transceivers. 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.

5.2.1. 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.

```
./asyncc2c 0 0 -i
./asyncc2c 0 0 -e
./asyncc2c 1 2
```

5.2.2. 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.

```
./asyncc2c 0 0 -I
./asyncc2c 0 0 -e
./asyncc2c 1 2 -I
./asyncc2c 0 0 -I -X
./asyncc2c 0 0 -e -X
./asyncc2c 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 field to the `sio4_async_t` structure prior to calling `sio4_async_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.

5.3. Debugging Aids

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

5.3.1. 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	OS
Application	`id`	`…/id/`	Linux
	`id.rta`	`…\id\`	INtime

5.3.2. sio4_async_show()

The function `sio4_async_show()` (section 4.1.9, page 14) is part of the protocol library interface. The purpose of the function is to produce a human readable report of all fields included in the `sio4_async_t` structure (section 4.3.1, page 16) passed in as a function argument. The function is best used to report the structure's content before it is passed to `sio4_async_set()` (section 4.1.8, page 14) or after it is passed to `sio4_async_get()` (section 4.1.2, page 10). The output can be used with Figure 2 to help visualize the channel configuration reflected by the structure content. When used in conjunction with `sio4_async_set()`, the `sio4_async_show()` output indicates the state that `sio4_async_set()` is expected to produce. When used in conjunction with `sio4_async_get()`, the `sio4_async_show()` output indicates the channel's current state. This may be beneficial after calling `sio4_async_set()` in order to verify the results achieved. The pair of calls may also be used before or after `read()` or `write()` calls in order to help explain the results of individual transfer requests.

5.3.3. 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_async_read()` or `sio4_async_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 by calling `sio4_async_open()` (section 4.1.6, page 12).
`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`	If non-zero, then the output will include a dump of all `GSC_USC_xxx` registers. Refer to `sio4_usc.h` for a complete list of these registers.
`usc_detail`	If non-zero, then the dump of the USC registers will include detailed information about all register fields, including the field value and the meaning of the value.

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

5.3.4. Status Return Values

The Asynchronous 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.

> **NOTE:** When an error status is returned by the Asynchronous Init Data, Set, Get or Show functions, the `err` argument to these same functions provides a string that identifies the offending argument or structure field.

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 SIO4 experienced an Rx FIFO Overrun, which means the application isn't reading data fast enough to prevent an overrun. This can also occur with an Rx FIFO underrun or a Tx FIFO overrun, but generally only with dedicated tests.
-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.
-22	-EINVAL	A function argument or referenced structure field value is invalid.
-71	-EPROTO	The Asynchronous Protocol Library has not been initialized. Applications must call `sio4_async_init()` 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_async_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 any of the functions listed in the above note is called on an SIO4 that is not based on the Zilog Z16C30 DUART. The board is either a -SYNC mode or a model with custom firmware.

5.4. Clocking Configurations

The Asynchronous serial protocol requires an Rx Clock signal for data reception. This limits the clocking configuration options when using the SIO4 for full duplex data transfer. The SIO4 receiver clocks in data from the cable's Tx Data signal using the cable's Rx Clock signal. In this case both signals are routed directly from the cable interface to the USC. The SIO4 transmitter clocks out data to the cable's Tx Data signal using the clock driven on the cable's Tx Clock signal. In this case the origin of the cable's Tx Clock signal and the USC's transmit clock is the SIO4's programmable oscillator, which is programmed to the desired Tx bit rate. The Tx Data is routed directly from the USC to the cable interface. Clock and data signal routing for SIO4B and later model boards is illustrated in Figure 3. This signal routing pictured is the configuration produced by the `sio4_async_init_data()` function, which initializes the `sio4_async_t` structure given as an argument.

> **NOTE:** On SIO4A model boards, full duplex operation requires either that the driver be updated to support programming of the SIO4A's programmable oscillator or an alternate signal routing configuration which may require an Rx Clock cable signal that is active full time.

NOTE: On SIO4 model boards (with no 'A' or 'B' suffix) full duplex operation requires either that the board's fixed frequency oscillator be replaced to match the desired transmit bit rate or an alternate signal routing configuration which may require an Rx Clock cable signal that is active full time.

The below code sample illustrates the minimum required steps to configure an SIO4 for Asynchronous operation. Error checking is omitted for brevity.

```
void config_sample(int fd)
{
    const char*          err     = NULL;
    sio4_async_init_t    init;
    sio4_async_t         async;

    init.tx_bit_rate     = 1000000L;
    init.rx_bit_rate     = 1000000L;
    sio4_async_init_data(fd, &init, &async, &err);
    sio4_async_set(fd, &async, &err);
}
```

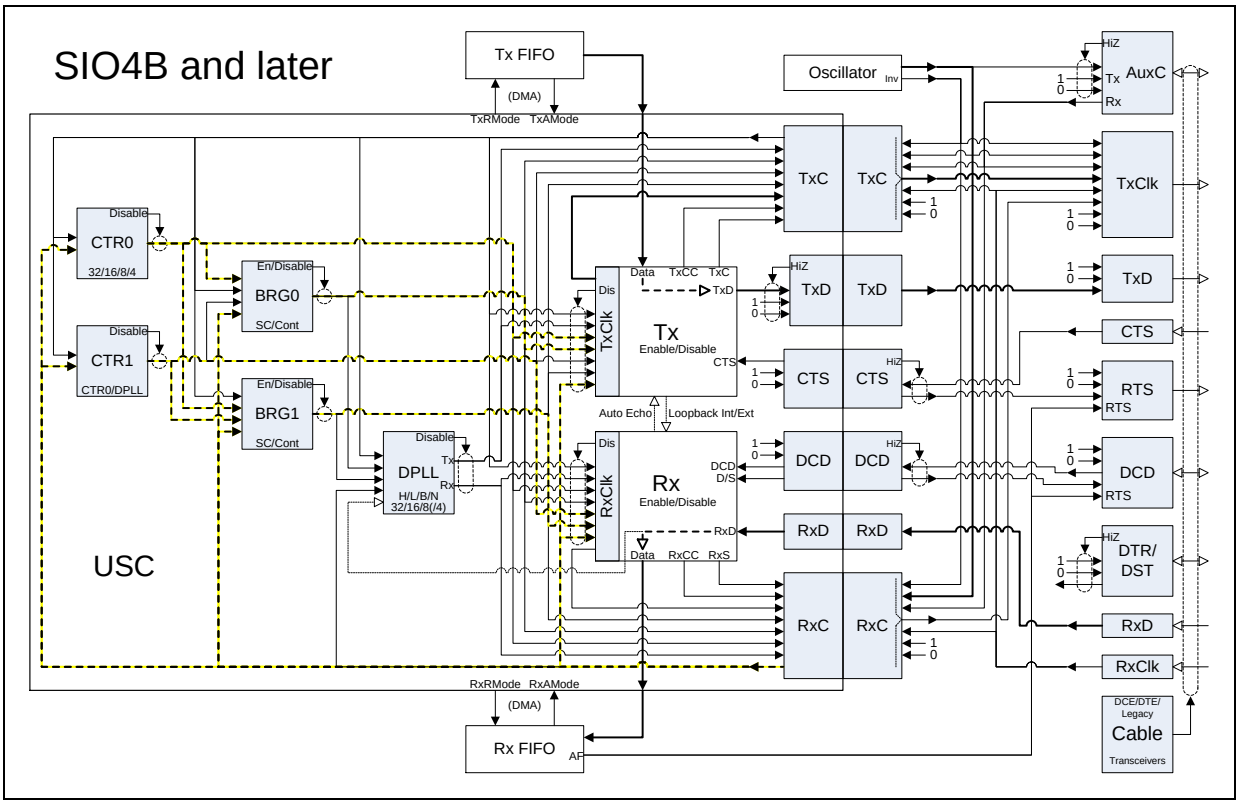

Figure 3 This illustrates the default Asynchronous clock routing on SIO4B and later model boards.

5.5. Cable Configuration Modes

The SIO4 supports two cable interfacing modes; DCE/DTE mode and Legacy mode. Older boards support only Legacy mode. More recent boards support only DCE/DTE mode. Intermediate boards support both modes. When both are available selection of the active mode is governed by enabling or disabling the transceivers. This is done through the `sio4_async_t.cable.enable` field, which is described under section 4.3.1.2 beginning on page 20.

5.5.1. DCE/DTE Mode

The DCE/DTE Mode controls cable signaling according to the DCE or DTE selection, as described in the board user manual. When available in firmware this mode is enabled by enabling the cable transceivers (see paragraph above). The Asynchronous Protocol Library passes all DCE/DTE mode settings to the driver unconditionally. DCE/DTE mode settings received by the driver are applied only if this mode is supported by the board. The driver otherwise ignores these settings.

5.5.2. Legacy Mode

The Legacy More of operation controls signal routing for the cable interface, according to the Upper and Lower settings, and the USC, as described in the board user manual. When available in firmware this is the default mode of operation. When selectable this mode is activated by disabling the cable transceivers (see paragraph above). The Asynchronous Protocol Library's `sio4_async_set()` function (section 4.1.8, page 14) applies Legacy mode settings only if the Legacy mode will be active when the function exits. The library otherwise ignores the Legacy settings. All other library functions process their settings unconditionally. Legacy mode settings received by the driver are applied only if this mode is supported by the board. The driver otherwise ignores these settings.

5.6. Error and Status Detection

The serial controller used on the SIO4 incorporates the ability to detect a number of error and other conditions for both the transmit and the receive data streams.

5.6.1. Interrupt Events

The most efficient means of detecting 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 interrupt event. (See the Interrupt and the Wait Event services in the driver reference manual.) This is illustrated in the following code fragments.

Thread A	**Thread B**

`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.`
`  }`
`  …`
`}`
	`For (;;)`
	`{`
	`  …`
	`  Enable desired interrupts.`
	`  Wait for an interrupt.`
	
	`  if (error interrupt occurred)`
	`  {`
	`  Record the error.`
	`  }`
	`  …`
	`}`

5.6.2. Rx Status Word

The SIO4 can also provide status in the Rx data stream on a per byte basis. This is done by enabling the Rx Status Word feature (`sio4_async_t.rx.status_word`, section 4.3.1.4, page 27). When enabled, the SIO4 places the lower eight bits of the USC's Receive Command/Status Register in the Rx FIFO immediately after the data

itself. This allows an application to identity the precise location in the data stream where some Rx related conditions occur. The downside of this is that it doubles the volume of data going through the Rx FIFO and effectively reduces its size by 1/2. Refer to the *Z16C30 Data Handbook* for information on the USC's Receive Command/Status Register.

5.7. Exclusions

5.7.1. Global Rx FIFO Full Configuration

The global Rx FIFO Full Configuration setting (see `SIO4_IOCTL_RX_FIFO_FULL_CFG_GLB` in `sio4.h`) is not included as part of the Asynchronous Protocol Library. It is excluded because the setting can override the channel specific settings for all four channels. If an application is to access this feature it must be done in parallel with use of the Asynchronous Protocol Library.

Document History

Revision	Description
March 19, 2024	Updated release date. Added comment about use of the static libraries. Removed the Cable Protocol Disable option as it is unsupported by hardware.
November 16, 2023	Updated release date.
June 15, 2023	Updated release date. Minor editorial updates.
December 13, 2022	Updated release date. Minor editorial updates.
June 2, 2022	Updated release date. Updated the description of the Tx Idle Line conditions.
February 8, 2022	Updated release date. Updated the description and calculation for the field `osc_prog`. Added the `file` argument to the `sio4_async_show()` service. Reorganized the Operating Information section (section 5, page 38). Added a Getting Started section (section 5.2, page 38). Added information on common error values returned by the Library. Minor editorial modifications.
August 9, 2021	Updated release date. Updated the I/O timeout field descriptions. Updated the descriptions of the I/O return values. Changed some argument and field names for consistency and clarity. Added note about read requests being serialized. Added note about write requests being serialized. Added statements about the read and write services being blocking calls.
May 5, 2021	Updated release date.
February 26, 2021	Updated release date.
February 18, 2021	Updated release date.
October 12, 2020	Updated release date.
October 12, 2020	Updated release date.
July 30, 2019	Updated release date.
March 24, 2019	Updated release date.
March 15, 2019	Updated release date.
October 17, 2018	Updated the default PIO threshold value based on testing. Updated the inside cover page. Added Tx and Rx I/O DMA Threshold fields.
October 31, 2017	Added information on the Cable Configuration Modes. Updated some legacy setting information.
October 17, 2017	Removed library versioning. Added information on opening device index -1. Added section on library interface files. Updated return status information for high level services. Added support for device index -1. Numerous editorial changes throughout.
December 7, 2016	Updated the operating information section. Made various miscellaneous updates. Updated information on loopback mode.
September 16, 2016	Updated to version 1.3. Updated the description of the `init.osc_prog` field. Changed the `cable.dcd` default for consistency with other libraries. Renamed the DPLL Rate macro from `…_RATE_4X` to `…_RATE_CTR1_4X`.
April 13, 2016	Updated to version 1.2. Updated the description of the `init.osc_prog` field.
September 7, 2015	Updated to version 1.1.
December 9, 2014	Initial release, version 1.0.